Design and Analysis of Machine Learning Tools for Segregating Plants in Rainforests

Alex Bzdel, Rucha Patil

Table of Contents

I. Problem Description

Species identification is a crucial process of foliage mapping. Understanding the natural world and the interactions between various animals requires a thorough grasp of biodiversity. For recording and comprehending the diversity of life on Earth, accurate species identification is essential. Proper species identification also helps conservationists to take the required precautions to conserve and maintain the habitats of these endangered or vulnerable species.

Today, there is a vast amount of foliage on the earth and a large portion of it remains undiscovered. And this knowledge can aid in conservation efforts aimed at preserving the forest and its inhabitants as well as offer important insights into the biodiversity and ecology of the forest. Manual exploration is not efficient, and also possesses safety concerns. As the people might get attacked or encounter poisonous species. Hence mapping the forest flora using a drone is a significant development.

Drones are able to scan larger areas in short time, record high-quality imagery and are more cost effective than manual exploration. Compared to conventional ground-based techniques, this can aid in the quicker identification of species distributions. Due to isolated locations, many species might not have been previously recorded. Sensors aboard drones may be used to track forest conditions in real time, including temperature, humidity, and air quality. Species that are particularly vulnerable to environmental changes, such as those endangered by climate change or habitat loss, can be identified using this information.

However, drones come with their own challenges. Drones have a limited field of vision, and it is difficult to distinguish species with similar physical characteristics. The lack of training data can be a significant challenge. Developing accurate algorithms for species identification requires a large and diverse dataset for training, and obtaining such data can be difficult in some areas. Some species may have subtle differences in color or reflectance that are difficult to capture using standard camera technology. Processing the huge amount of acquired data is a challenging task. Especially when we are not aware of what specifically we are looking for and searching in an ocean of data of same looking images.

In this class we as a part of the Species Identification team attempted to identify plant species from the high-quality videos captured by the XPRIZE team. We present the methodology and an analysis of different techniques we tried and implemented to address some of the issues mentioned above.

For the task of species identification, we focused our project on the application of XPRIZE. What we attempt to achieve is the isolation of unique plant species from the video footage and generating images of the same. These images can then be further processed to run identification algorithms or be directly uploaded to platforms like iNaturalist for experts to identify. We have implemented 5 methods for this purpose.

1. Stem Detection using Perceptree
2. WebODM 3D Forest Reconstruction
3. Image Color Segmentation
4. Depth Detection
5. Image Segmentation Using Graphs

We attempt to detect the prominent trees using the tree stems. Observing prior successful identification in the iNaturalist, we noticed that some trees were identified using stems. Hence, we explored this as the first method. The second step was to identify the shrubs and isolate clear looking leaf clusters. We implemented the third of Image color segmentation for this purpose. We utilized the K-means clustering method for the same. The goal of using depth detection in our software was to be able to systematically determine whether tree trunks or plants were blocking one another. We achieved this using the Monodepth2 algorithm. We attempted the depth perception using the WebODM software priorly. However, it was extremely time-consuming and didn't yield usable results. We also tried to segment the image using the graph based clustering algorithm for image segmentation as a comparison for the k means clustering.

We present the methods in two sections below which are Methodology and Experiments. The Methodology explains the working of the algorithm and procedure we followed. The experiments section focuses on the results and analysis.

II. Introduction

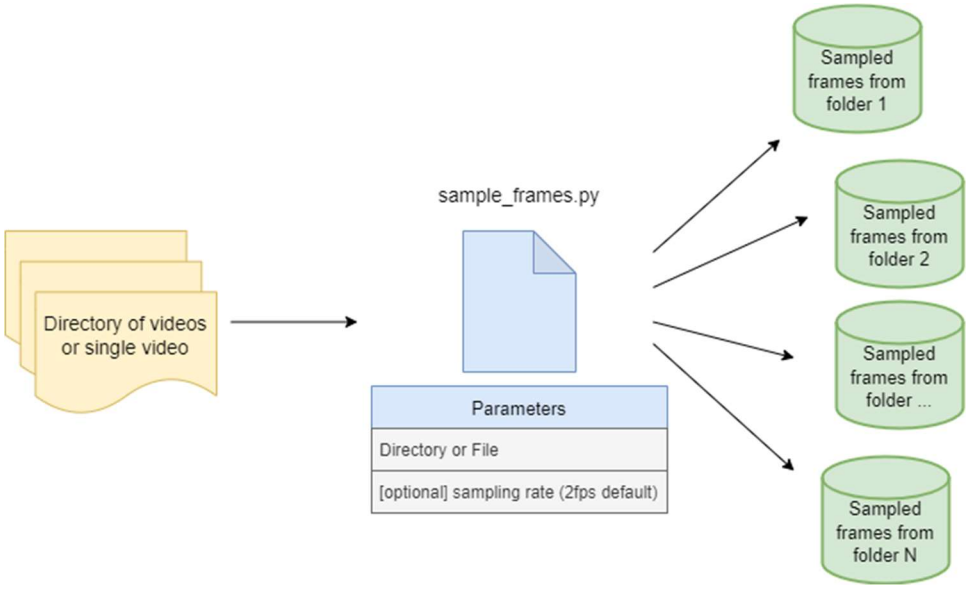

Figure 1. Architectural flowchart of frame sampling software. A video or directory is passed to the software and frames are automatically sampled accordingly.

Getting accurate images of rainforest wildlife and plants is extremely important for researchers looking to study our planet's ecosystem today. Drone pilots no longer must focus on taking single pictures throughout their flights but can leave the camera running so pictures can be sampled from them later. Manually sampling pictures from videos has been the best option for a long time, but the advent of better and cheaper computing power and higher resolution drones in the past 10-15 years have enabled this image sampling to be done algorithmically (Klose et al., 2015)

Our team looked to contribute to the advancements of algorithmically determining which images are worthy of researchers' time through our Rainforest Engineering course at Duke University. We experimented for part of our semester before amalgamating our work into one final command line system that researchers can use to sample better images from their videos at a fast rate of speed.

Our program works by taking in a single video or a directory of videos. While this software was made with drone forest videos in mind, the software can accept any type of video (albeit it

may not work as well as the forest examples we tested on). The sampling rate can be adjusted by adjusting the program internally but is not malleable through the command line itself, but we set it to two frames per second by default. Once this program is run on a folder or video, the software creates another directory that houses the frames that are sampled from the video (or each video). Then, these frames are passed to another program which either detects the stems in a frame or performs segmentation to pick out unique objects.

These steps are taken with the goal of providing researchers with a better alternative to manually watching videos and saving pictures whenever they see fit. Through our experimentation and results, we believe that we developed a program that can be beneficial to those studying our environment as well as future researchers who would like to improve upon the limitations of our methods.

III. Methodology:

III.I Method 1 – Stem Detection

III.I.I Data and Pre-processing

One of the biggest challenges in the forestry and species identification is the unavailability of Data. One current suggested method to overcome this challenge is to use synthetic datasets. However, annotation of the dataset is also challenging. There aren't many datasets that are specifically for forestry, which restricts deep learning applications and task automation needing high-level cognition. The paper [1] makes use of Unity Game engine to generate a model of the forest along with annotations. The authors generated a dataset of over 43000 images using this technique and named this dataset as the SYNTHTREE43k dataset. Based on this dataset Mask R-CNN was trained to generate the weights.

The Unity game engine and Gaia2 were used to build the new dataset SYNTHTREE43K, which contains more than 43,000 RGB and depth photos and more than 162,000 annotated trees. Realistic tree models from Nature Manufacture were used to create the virtual woods, and several spawn rules were used to regulate the density of the forest. Procedural terraforming and terrain texturing were used to build the area, and additional realistic elements like stumps, grass,

and bushes were added. The snow, wet effects, and particle systems that created snowflakes, droplets, and fog effects were used to replicate the weather conditions.

The lighting was altered to represent various times of day, creating accurate shadows on the scene's items. Five key points are allocated to each tree, and each picture in the collection has a bounding box, segmentation mask, and key point annotations. The pipeline can produce an infinite number of synthetic pictures at a pace of 20 frames per minute for annotation. The production of the dataset enables, among other forestry-related applications, the development and training of autonomous tree-felling systems.

III.I.II The Algorithm and Model

The Mask R-CNN architecture is composed of [1,2]

1. A convolutional feature extraction backbone,
2. A Region Proposal Network (RPN)
3. Prediction heads

The original model was slightly modified to adapt to the problem. The key point branch (Which is optional was added). This allows the model to be used for classification, bounding box regression, segmentation, and key point prediction.

Mask R-CNN makes predictions as a two step process.

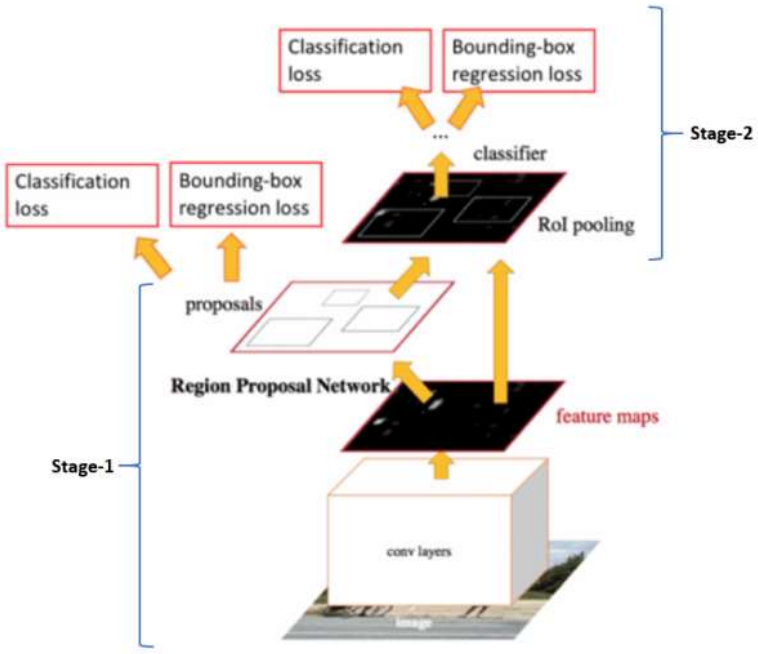

Figure 2. Mask RCNN

Stage I : The first stage has two networks, backbone, and the regional proposal network. Regional proposals are the areas where the object of our interest exists. The model we adopted made use of three backbone architectures ResNet-50, ResNet-101, and ResNeXt-101. Resnet gives excellent gains in both accuracy and speed for feature extraction [1]. The region proposal network generates regions of interest (RoI) from the feature maps of the backbone using default nine box anchors, corresponding to three area-scales (8, 16 and 32) and three aspect ratios (0.5, 1.0, and 2.0).

Stage II : The network determines the object class and bounding boxes for each suggested region. To create predictions, completely linked layers need a fixed-size vector, whereas each suggested area might be of a varied size. Generally the RoI pool technique or the RoIAlign method are used to set the size. RoIAlign uses bilinear interpolation to map the feature maps of the backbone into a 7 by 7 input feature map inside each RoI area. The network head then processes the features from each RoI to forecast the class, box offset, binary segmentation mask, and optional binary mask for each key point concurrently.

TABLE I: Backbone parameters. The number of learnable parameters (#Params), computational complexity (GFLOPs) and frames per second (FPS) at inference time on 800 × 800 images.

Backbone	#Params	GFLOPs	FPS
ResNet-50-FPN	25.6 M	3.86	18
ResNet-101-FPN	44.7 M	7.58	15
ResNeXt-101-FPN	44 M	7.99	10

III.I.III Training Details

SYNTHTREE43K is divided into three subsets for model training: 40 k for the train set, 1 k for the validation set, and 2 k for the test set. A stochastic gradient descent (SGD) optimizer with a momentum of 0.9 and a weight decay of 0.0005 is used by the model to learn from the train set. Gray scales of an 8-bit 1-channel make up depth pictures. They are transformed from pre-trained models at train time into 8-bit 3-channel to match the RGB format. Only for ResNet-50-FPN are hyperparameters optimized, and these hyperparameters are applied to all models.

III.II Method 2 – WebODM 3D Forest Reconstruction

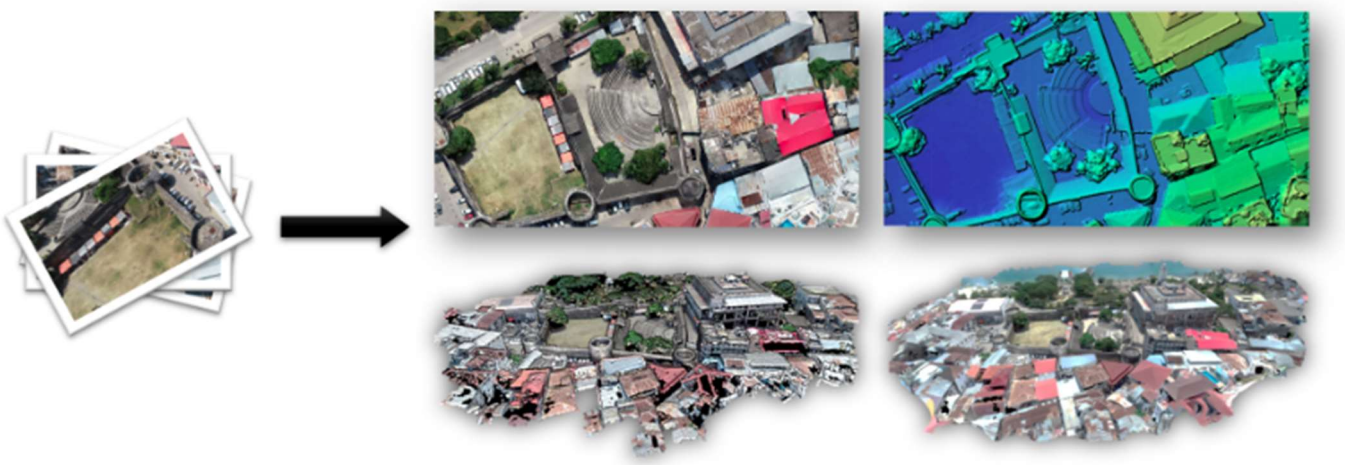

Figure 3. WebODM diagram showing the desired effects of its software. A set of images are passed in and, based on the overlapping areas, a 3D environment can be reconstructed (OpenDroneMap Authors).

Our team decided to use 3D forest reconstruction to pick out important objects in a set of forest images. There are many 3D reconstruction methods, but we decided WebODM would be best due to its specific application to drone-based videos (Ham, Wesley, Hendra, 2019).

WebODM is an open-source photogrammetry software that uses multiple images to generate a 3D model of a landscape or object (OpenDroneMap Authors). In our case, we used WebODM to attempt to reconstruct 3D environments from sampled frames from drone forest videos. The software processes each frame by identifying matching points across the images and then uses these points to create a point cloud representing the surface of the object being reconstructed. This point cloud is then converted into a 3D mesh, and finally, a texture is applied to the mesh to create a realistic representation of the object.

We tested four different methods in order to achieve this goal. Specifically, we attempted to reconstruct the 3D environment using one frame per second sampled for 30 seconds, two frames every second for 30 seconds, one frame every second for 10 seconds, and two frames every second for an entire minute-long video.

Despite our efforts, the results were not promising for 3D reconstruction purposes. Although we varied the sampling rate and duration of the video frames, the resulting reconstructions were not accurate representations of the original forest environment. This failure may be attributed to the complexity and unique characteristics of the forest environment, such as the varying depths of foliage and the movement of the drone. Our “Experiments and Evaluations of Methods” section contains more details on the process of coming to this conclusion.

III.III Method 3 – Image Color Segmentation

Image color segmentation is used to help us with leaf identification in our images. Leaves contrast heavily with tree trunks and branches, and oftentimes with leaves of other plants. With this method, we sought to segment out different plants and objects in an image by utilizing the color diversity that naturally exists in the forest. While this step includes simpler methods (like K-means), it yielded arguably our best results.

According to “Review on Determining of Cluster in K-means clustering”, written by Kodinariya and Makwana (2013), K-Means clustering is a commonly used method for clustering data into groups based on similarities between data points. In our case, we wanted to use this method for image color segmentation. Effectively, we used this process to group similar pixels together based on their color values, resulting in segmented regions with similar colors. This helped us segment out major objects in a sampled video frame and determine whether the image would be useful for researchers.

To implement the K-means algorithm for image color segmentation, we first convert an image into a matrix of pixel color values, where each pixel is represented by a three-dimensional vector of red, green, and blue (RGB) color values. We then use the K-means algorithm to group similar pixels together based on their RGB values, resulting in a set of clusters that represent different color regions in the image.

One must note that the number of clusters used in the algorithm will affect the quality and accuracy of the resulting segmentation. To determine the optimal number of clusters, Kodinariya and Makwana suggest using the "elbow method," where the within-cluster sum of

squares (WCSS) is plotted against the number of clusters used. The "elbow" of the plot represents the optimal number of clusters to use, where increasing the number of clusters beyond this point does not significantly improve the quality of the segmentation.

While the elbow method is a useful technique for determining the optimal number of clusters in K-means clustering, we ultimately decided not to use it in our implementation. This was mainly due to the increase in computational time required to run the code, as the elbow method requires the K-means algorithm to be run multiple times for different numbers of clusters. Instead, we opted to manually choose the number of clusters based on visual inspection of the resulting segmentation. While this approach may not be as precise as the elbow method, it allowed us to achieve satisfactory results while minimizing computational time. Through a process of manual optimization, we determined that setting the number of clusters to three generally yielded good results.

After obtaining the mean colors for each cluster, the original image is copied to create a new image to which the segmented objects will be added (Kodinariya, Makwana, 2013). For each cluster, a masked image is created by drawing the segmented objects using the OpenCV **drawContours()** function. For the purposes of plotting each cluster along with the original image, these masked images are concatenated horizontally using the OpenCV **hconcat()** function to create the final segmented image. The image is then converted to grayscale using the OpenCV **cvtColor()** function, followed by histogram equalization using the **equalizeHist()** function to increase contrast (Kodinariya, Makwana, 2013)

The mean pixel value of the equalized cluster is then calculated using NumPy's **mean()** function. If the mean pixel value is above a threshold of 50, the image is considered ‘good’, otherwise, it is considered ‘not good’. The threshold value of 50 was determined through another process of manual optimization in which we looked at the quality of various results of segmentations at different values.

If an image has a single cluster considered ‘good’, we save the image along with all associated clusters to our local filesystem. Its filename references both the video it came from and the specific frame it was sampled from so that users of the software can easily trace the

origins of each image and its clusters. If all clusters of an image are considered ‘not good’, we conclude that the image does not have any qualities that would make it a good candidate for further inspection by researchers.

Depending on a user’s tolerance for having enough ‘good' images, the mean value of 50 can be adjusted. If changed to a lower value, more images will be considered good and, therefore, more images will be saved. This, of course, leads to an increased workload for the human at the end of the process checking which images we saved that should be sent on for further evaluation. Alternatively, if the mean value threshold is moved above 50, fewer images will be saved. The tradeoff in this case is that a greater number of truly good images may be ignored by the algorithm, but the human at the end of the process has less images to look through.

While color segmentation based on mean color is a simple and effective method, it has limitations when it comes to detecting ‘good’ objects, since color diversity is not the only indicator of a good object. Other characteristics such as texture, shape, and size can also be important factors to consider. In future work, it may be beneficial to explore other segmentation methods that consider these additional features.

There are a variety of alternative methods that could be explored in future work for improving color segmentation beyond just mean color. One approach involves combining color segmentation with other image processing techniques such as edge detection or texture analysis to produce more accurate segmentations. Additionally, deep learning approaches, such as convolutional neural networks, could show promise in image segmentation tasks and could be a viable alternative to traditional clustering or color-based methods. Further investigation and experimentation with these techniques may lead to more robust and accurate color segmentation methods for a variety of applications.

Our segmentation code used for this part of the analysis can be found at github.com/abzdel/rainforest_image_analysis (will be updated before final draft).

III.IV Method 4 – Depth Detection

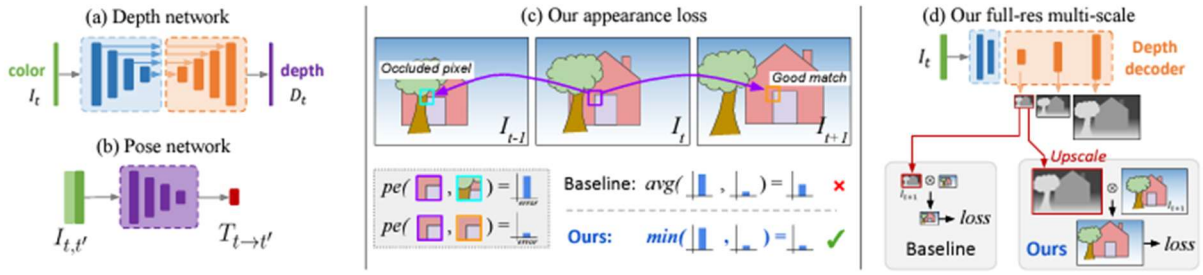

Figure 4. Monodepth2 Depth Network diagram (Godard, Mac Aodha, Firman, Brostow, 2019).

The goal of using depth detection in our software was to be able to systematically determine whether tree trunks or plants were blocking one another. Forest images are extremely noisy, so making this distinction could work in conjunction with stem detection or image color segmentation to ensure more of our results are correct. The goal was to produce a depth map for each of our images. There are many depth detection algorithms used by researchers today, but we decided on Monodepth2 as it is a state-of-the-art method that outperforms every other depth detector (Godard, Mac Aodha, Firman, Brostow, 2019).

Additionally, there are many methods of performing depth detection. One method we considered involved using multiple images of the same object (for example, a few frames sampled from a drone video a few seconds apart) and converging them to find the depth. However, using multiple images with WebODM caused an increase in runtime that rendered the program unusable for our purposes. Instead, we chose to use a single image to find the depth.

Monodepth2 is an open-source framework developed by Niantic Labs for depth estimation from a single monocular image (Godard, Mac Aodha, Firman, Brostow, 2019). The program uses a neural network that takes an input image and predicts the corresponding depth map. The depth map provides information about the distances of different objects in the scene from the camera.

The Monodepth2 program is based on the use of a fully convolutional encoder-decoder architecture that uses residual connections to better propagate gradients through the network. The encoder network takes the input image and reduces its spatial resolution while increasing

the number of feature channels. The decoder network then upsamples the feature map back to the original resolution and generates the corresponding depth map.

To train the network, Monodepth2 uses a combination of stereo images and monocular images with known depth. The loss function used to train the network is a combination of a depth regression loss and a depth smoothness loss, which encourages the depth map to be both accurate and smooth. The pretrained model used in this program, mono_640x192, utilizes 3,152,724 parameters.

It's important to note that while Monodepth2 can provide accurate depth estimates in many cases, it is not perfect and may not work well in all situations. Factors such as lighting conditions, image quality, and scene complexity can all affect the accuracy of the depth estimates. However, Monodepth2 can be a useful tool for many image processing and computer vision applications (Godard, Mac Aodha, Firman, Brostow, 2019).

The Niantic Labs GitHub repository can be found at github.com/nianticlabs/monodepth2. Our team also has a repository that we used for the testing of our images, which can be found at github.com/abzdel/monodepth_rainforest.

III.V Method 5 – Graph based Clustering for Image Segmentation

Graph-based image segmentation is a popular approach to segmenting images into regions with similar characteristics. The basic idea behind graph-based segmentation is to represent the image as a graph, where the nodes represent the pixels or image patches and the edges represent the similarity or dissimilarity between them. The graph is then partitioned into disjoint subgraphs or clusters, where each cluster represents a distinct region in the image. Graph-based segmentation methods can use different similarity measures, such as color, texture, or gradient, and various algorithms, such as minimum spanning tree, normalized cut, or spectral clustering, to partition the graph.

Efficient Graph-Based Image Segmentation (EGBIS) is a widely used algorithm for image segmentation that has several advantages and disadvantages. One of the main advantages of EGBIS is its efficiency. The algorithm is able to segment large images in a short amount of time,

making it ideal for real-time applications. EGBIS also produces accurate segmentation results, particularly for images with complex structures and textures. It is a flexible algorithm that can be used with different types of image features, and it is robust in handling noise and other artifacts in the image. However, EGBIS also has some limitations, such as the need for parameter tuning, the risk of over-segmentation, and the potential loss of detail in the boundary smoothing process. Additionally, EGBIS requires a large amount of memory, which can be a limitation for resource-limited systems or large images. In summary, EGBIS is a fast and accurate algorithm for image segmentation that has several advantages and limitations that need to be considered when choosing an appropriate segmentation method.

The Method that the algorithm follows is

1. Image preprocessing: The input image is first preprocessed to reduce noise and enhance edges. This can involve techniques like smoothing, gradient computation, and edge detection.
2. Building the graph: A graph is constructed where each pixel in the image represents a node, and the edges between nodes represent the similarity between pixels. The similarity measure can be based on color, texture, or any other feature that distinguishes one pixel from another.
3. Segmentation: The graph is segmented into disjoint subgraphs or clusters using a minimum spanning tree algorithm. The nodes are merged iteratively based on the weights of the edges connecting them, until a stopping criterion is met.
4. Post-processing: The resulting segments may contain small and noisy regions. Post-processing techniques such as thresholding, merging, and splitting can be applied to refine the segmentation.

IV. Experiments and Evaluation of Methods

IV.I Method 1 – Stem Detection - Experiments and Evaluation

IV.I.I Packages and Installations

We faced a lot of issues in the installation of the required packages due to compatibility issues. The detectron package which is required for the successful running of the algorithm cannot run on windows (due to one of it requirements failing), so we spent some time figuring out the package and CUDA issues. We were able to find out a workaround around that issue which required us to create multiple versions until we found a setting where everything worked with each other. It is as mentioned in the Document “Tree Logs”

IV.I.II Results

The tree stem detection algorithm was tested on a dataset of high-resolution aerial images of forests captured by the drone. The algorithm was able to accurately detect and segment the tree stems in the images with high precision and recall. Figure 5 and Figure 6 shows some sample results of the algorithm, where the green contours represent the ground-truth tree stems, and the red contours represent the detected tree stems. The algorithm was able to detect tree stems of different sizes and shapes and was fairly robust to noise and occlusion.

Figure 5. Tree Stem Detection 1.

Figure 6. Tree Stem Detection 3

However, as we observe in the above image, stems that are too close or two far have trouble being detected.

The accuracy of it being able to identify a stem in a low contrast environment was low. However, when the image was cropped we were able to detect the trees which were not detected in the previous run.

This leads us to believe that if we sample the images and iterate for blocks , we will be able to find the accurate number of tree stems.

Figure 7. Tree Stem Detection 3.

IV.II Method 2 – WebODM 3D Forest Reconstruction - Experiments and Evaluation

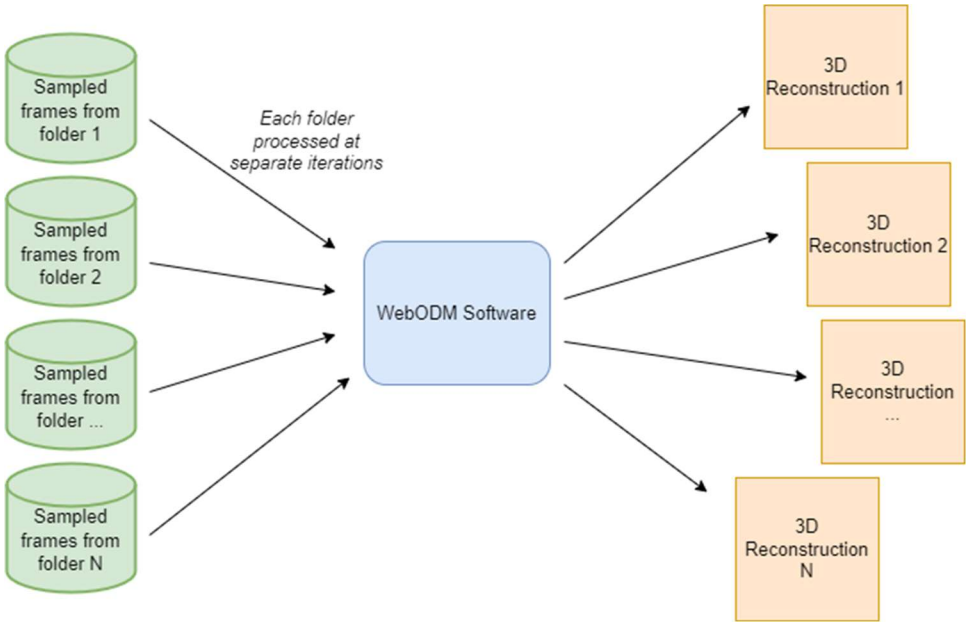

Figure 8. WebODM Image Pipeline. Each folder of sampled images is passed in (separately) to WebODM, resulting in 3D reconstructions for each.

Some of the resulting reconstructions from our WebODM pipeline can be seen below. Each example is from the same drone video panning from the base of a tree to the top. As stated in a previous section, we attempted to optimize this process with various sampling rates and number of frames captured.

One frame sampled every second for ten seconds:

Figure 9. 3D reconstruction with sampling rate of 1fps for ten seconds. The base of the tree trunk in the center and most of the skinnier tree on the left are able to be reconstructed.

While WebODM's algorithm can reconstruct major shapes within the images, too many details are left out for this to be a useful reconstruction.

One frame sampled every second for ten seconds:

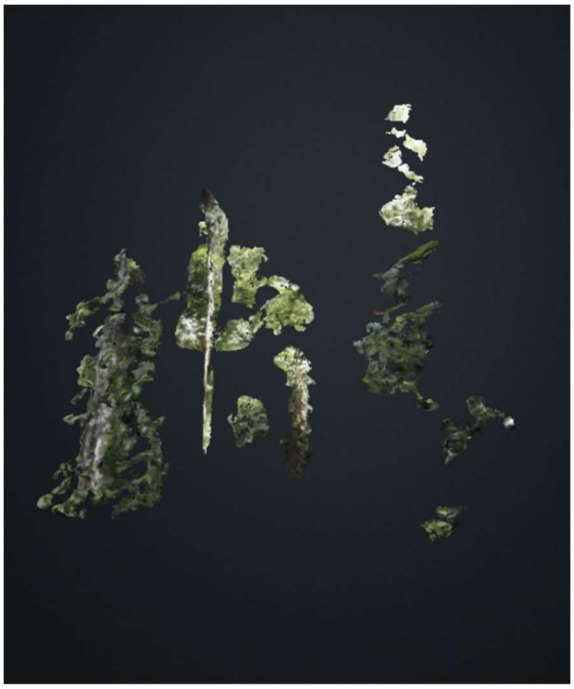

Figure 10. 3D reconstruction with sampling rate of 1fps for thirty seconds. This results in less detail and fewer relevant 3D objects.

Interestingly, increasing the number of frames captured while leaving the sampling rate constant yielded a much worse reconstruction of the environment. With this information in hand, we decided to increase the sampling rate to improve the result.

Two frames per second for 30 seconds:

Figure 11. 3D reconstruction with sampling rate of 2fps for thirty seconds. We see more detail in both the large tree trunk in the center and the skinnier tree on the left. We also see a reconstruction of one of the plants on the right.

With these parameters, some detail is lost, but we can see that more of each structure is reconstructed by WebODM. This was undoubtedly our most promising result.

We also attempted to pass the entire video (5 minutes, 31 seconds) through WebODM with a sampling rate of two frames per second. Sampling this large of a video yielded over 1300 frames to be processed. Even with the power of a GPU, the process failed after around 3 hours. WebODM does not give much information as to why its processes fail, but it is likely that the algorithm was having trouble connecting some of the frames together into one continuous image.

Additionally, our end goal was to make a tool that you could use with any drone video to ascertain a good result. Given how slow this process was with a GPU, it would not be a feasible solution to the problem we wanted to solve. Nevertheless, we believe that our methods may provide a useful starting point for future researchers looking to explore this area. We hope that our findings contribute to the future development of such tools.

IV.III Method 3 – Image Color Segmentation - Experiments and Evaluation

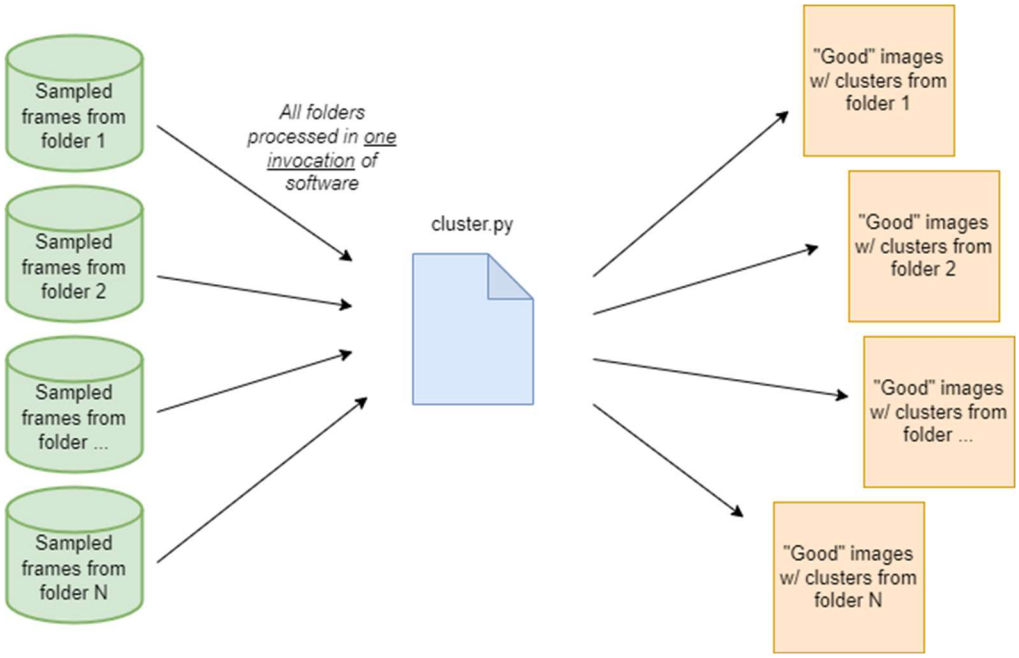

Figure 12. Image Color Segmentation Pipeline. All folders of frames are processed in one invocation of the software. The software then automatically sends good images to new local directories.

In this section, we present the evaluations and experiments we conducted using the color segmentation method to analyze our sampled images. We utilized this method exclusively to identify and isolate objects of interest within our images. Here, we provide an overview of the process and results of these experiments, along with our analysis of their implications for future research.

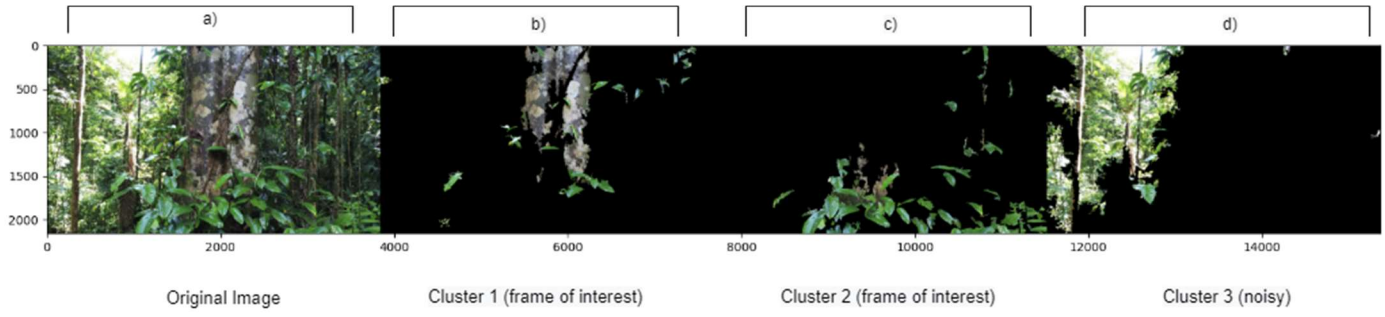

Figure 13. Each cluster is plotted next to the original image. In this example, clusters 1 and 2 indicate that we have a useful picture, so it would be sent to iNaturalist. Cluster 3 is just noise.

The pipeline laid out in Method 3 – Image Color Segmentation yields outputs in the format of figure 13. We also see an optional logging output that looks like the following:

mean value: 70.73 This is a good image.

mean value: 51.20 This is a good image.

mean value: 45.96 This is not a good image.

Each line above corresponds to its associated cluster from left to right, ignoring the first image in the plot (which is the original). In this case, line 1. above is associated with the first cluster from about 4000 to 8000 on the x-axis of the plot in figure 13. Line 2. references the second cluster from 8000 to 12000, and so on.

The text output of our program indicates that clusters 1 and 2 have promising results as there is a lot of color diversity in these segments, likely signifying an object of interest. While the objects are not made out perfectly, it is clear that segment 1 is marking the tree trunk and segment 2 is marking the plant at the base of the tree. Our algorithm correctly identifies these as objects of interest, and correctly identifies that the third cluster is mainly noise that we are not interested in. However, since a minimum of one cluster looks promising, we save this plot with the following format:

clustered_frames/segmented_{video_name}/{original_frame_number}_clusters.png

As stated previously, this helps the end user trace the origin of the video as well as the specific frame that was sampled. We have seen a lot of promising results using our algorithm.

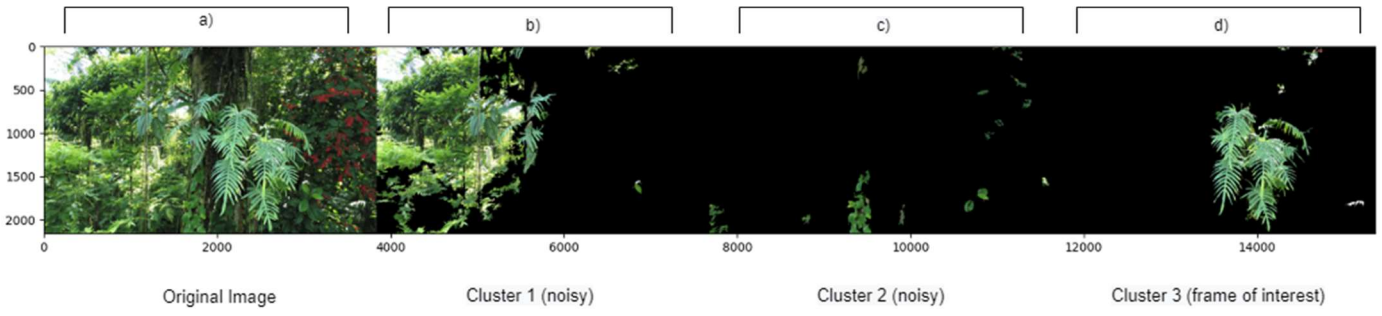

Figure 14. A sampled image of a plant in front of the tree. Cluster 3 shows that our algorithm can pick out the plant as an object of interest.

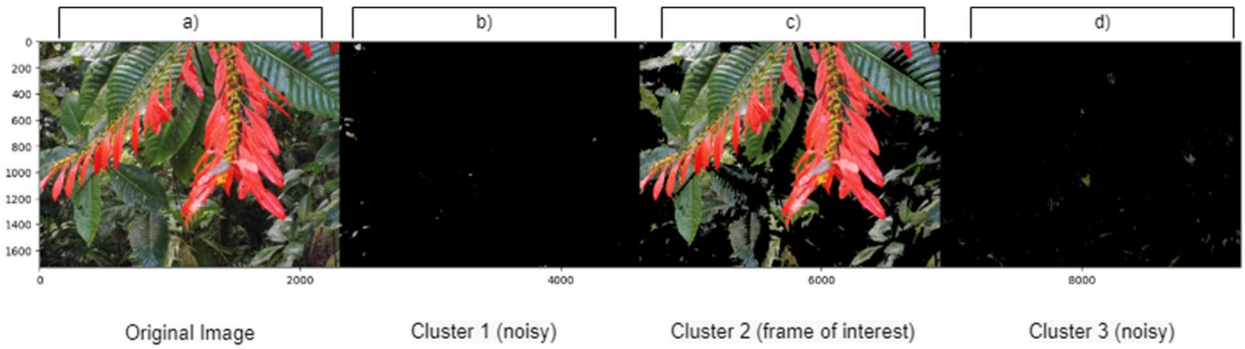

Figure 15. A sampled image of a colorful plant. This is an ideal candidate for our algorithm as the color of the plant is very diverse. Cluster 2 is picked as our frame of interest as the program has removed noise while preserving the shape of the object we want to see.

The process above is the best-case-scenario for our image clustering algorithm. We do, however, acknowledge the limitations that come with analyzing the result solely based on color diversity in an image.

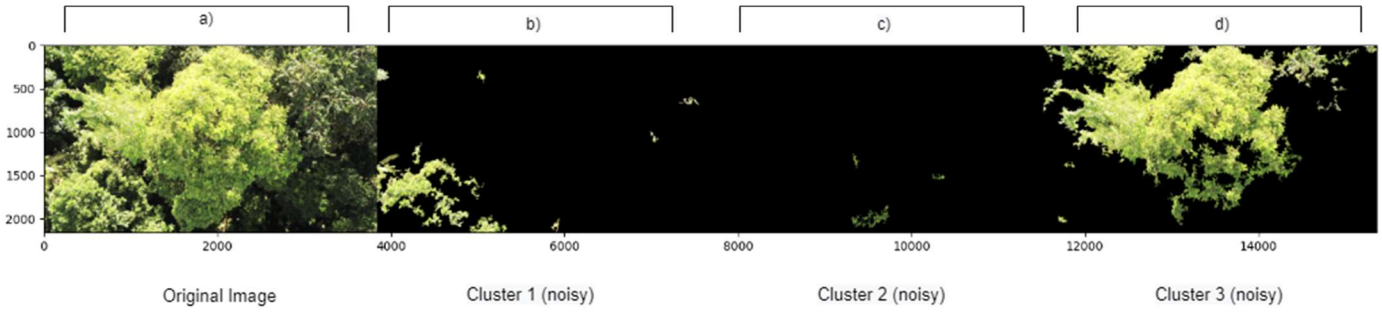

Figure 16. Top-down image of a tree. We would ideally like to see our algorithm eliminate this frame entirely, but the color diversity in the first cluster causes the software to mark this as a good image.

Another result that is problematic on multiple fronts can be seen in figure 16. First of all, this is not an image we would want to be sampled from a video in the first place. A top-down image such as this does not have enough valuable information to be sent to researchers for further evaluation. However, the software we have in place samples each type of frame equally. Adding another check here to make sure the initially sampled frames are something we want to look at is an obvious next step.

Additionally, there is not enough color diversity in the image to have meaningful color segmentation. However, our algorithm detects the following for each segment:

mean value: 67.48 This is a good image.

mean value: 45.21 This is not a good image.

mean value: 45.80 This is not a good image.

If our color segmentation step eliminated this image entirely, there would be no problem with the system. Since the first mean value is well above the threshold of 50, however, this image will be flagged for future inspection as a potential good image.

Therein lies a prime example of why using color segmentation as the only step is a very limited approach. In an image such as this, we have a very monochromatic histogram to start. Even if we equalize the histogram to better utilize the full range of the color spectrum, there are simply not enough different values for color segmentation to yield a relevant result (Wang, Zhang, 1999).

We believe that our method of sending our sampled frames through a color segmentation algorithm is a phenomenal starting point for this proof-of-concept system. Given more time, our team would like to explore how this color segmentation could act as a single marker of "goodness" for an image. We could also explore things such as how many black pixels are in each cluster, how many objects are detected in each cluster (using an edge detector), or how the interpolated texture of an object impacts the results. Using these markers in conjunction with one another could yield a much more robust system that builds on the strengths of the current program while minimizing the effects of its weaknesses.

IV.IV Method 4 – Depth Detection - Experiments and Evaluation

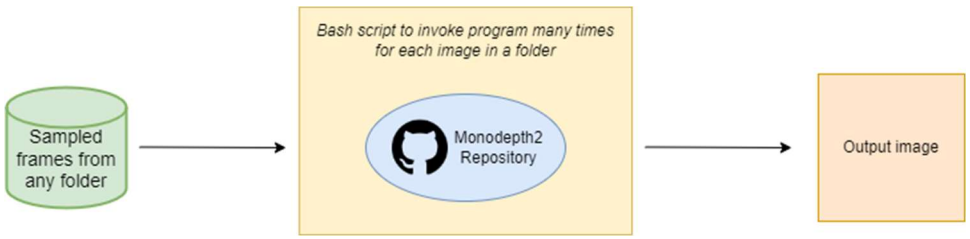

Figure 17. Monodepth2 custom implementation. We utilize the Niantic Labs Monodepth2 repository and wrap it in a bash script that can send multiple images to the model in one invocation of the software.

Monodepth2 did not work well for our purposes. We suspect that the quality and resolution of the input images may have affected the program's ability to accurately estimate depth. Additionally, the sampled forest images often had a lot of overlapping branches and foliage, which made it particularly challenging for the program to estimate depth in those areas.

We also considered that the specific settings and parameters used in the Monodepth2 program may not have been optimal for our images. For instance, tuning the number of layers in the encoder and decoder networks, adjusting the learning rate, and tweaking the weight given to different loss functions may have improved the performance of the program on our images.

In any case, it's important to note that the success of any depth estimation algorithm can be highly dependent on the specific context and conditions of the images being used. As such, we believe that it's crucial to experiment with different algorithms and parameters to find the best fit for a particular application.

Niantic Labs provides a sample image for depth estimation for users:

Figure 18. Sample image for depth estimated provided by Niantic Labs.

When running the monodepth2 program on this image, we see the following result:

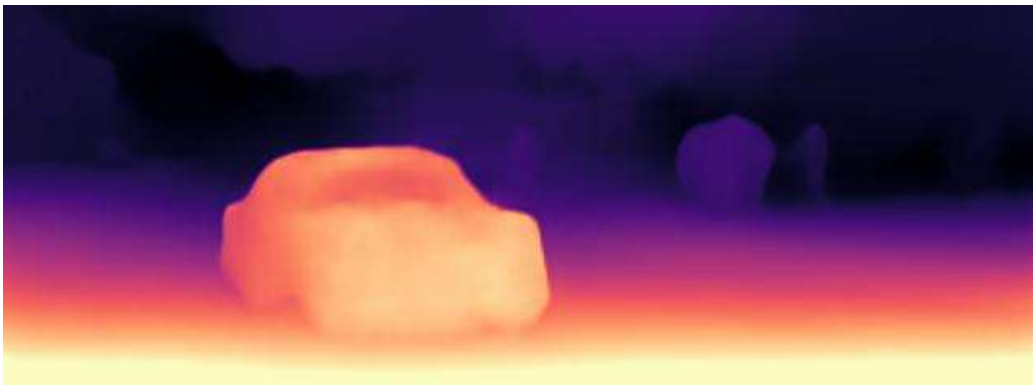

Figure 19. Result of sample image when passed through monodepth2 algorithm.

This is the optimal case for this depth estimator. As can be seen, however, the sample image in figure 19 is quite different from the forest images we have been sampling. In this case, we see a great result, but our images do not perform as well.

Figure 20. One of our images we sampled from a drone forest video.

Figure 20 shows a high-resolution image of one of the frames we sampled from a video. We passed this image through the monodepth2 algorithm and saw the following result:

Figure 21. Result of our forest image when passed through Monodepth2.

As seen in figure 21, the result for our image is not promising. The estimated depth map seems to be inaccurate, with several areas appearing to be either overestimated or underestimated in depth. While the training data for Monodepth2 is not made public, we assume that this poor result is due to the training dataset being vastly different from the images we are using. The forest environment is complex, with many overlapping objects and occlusions, which may have made it challenging for the algorithm to accurately estimate the depth of the scene.

Figure 22. Another sampled image from a drone video. A depth detector should accurately detect that the brush on the left side of the screen is behind the tree trunk.

Figure 23. Result from our new image. There is not much use for this result.

There are a variety of pre-trained models available for depth estimation, but none are specifically tuned to deal with images of our kind. As a next step, we believe future researchers could fine-tune one of these pre-trained models to make it more robust to forest images. By training the model on a dataset of forest images with known ground-truth depth maps, the model may be able to better capture the unique characteristics of forest scenes and improve its depth estimation performance.

Additionally, incorporating other visual cues, such as motion or texture, may also help improve the accuracy of depth estimation in forest scenes. We believe that there is significant potential for further research in this area, and we hope that our experience can contribute to the development of more effective depth estimation algorithms for forest environments.

IV.V Method 5

This method did not yield results as expected. Since the pictures were mostly covered with green, we believe the algorithm was unable to map the values appropriately which led to the resulting image being generated flat. For all the images that were passed to it, the resulting image kept being flat for all different kinds of input graph types as well(i.e grid, nn).

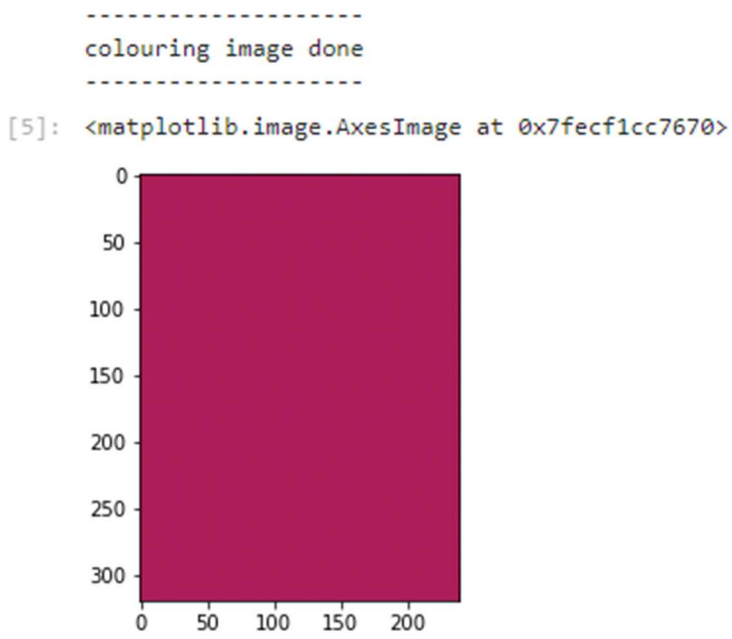

Figure 24. Results from graph based image segmentation

We think there could also be some issue with the tensors employed in the algorithm and further understanding of TensorFlow would be needed to write the program for our pipeline and make appropriate changes.

The algorithm was working well with the sample images provided

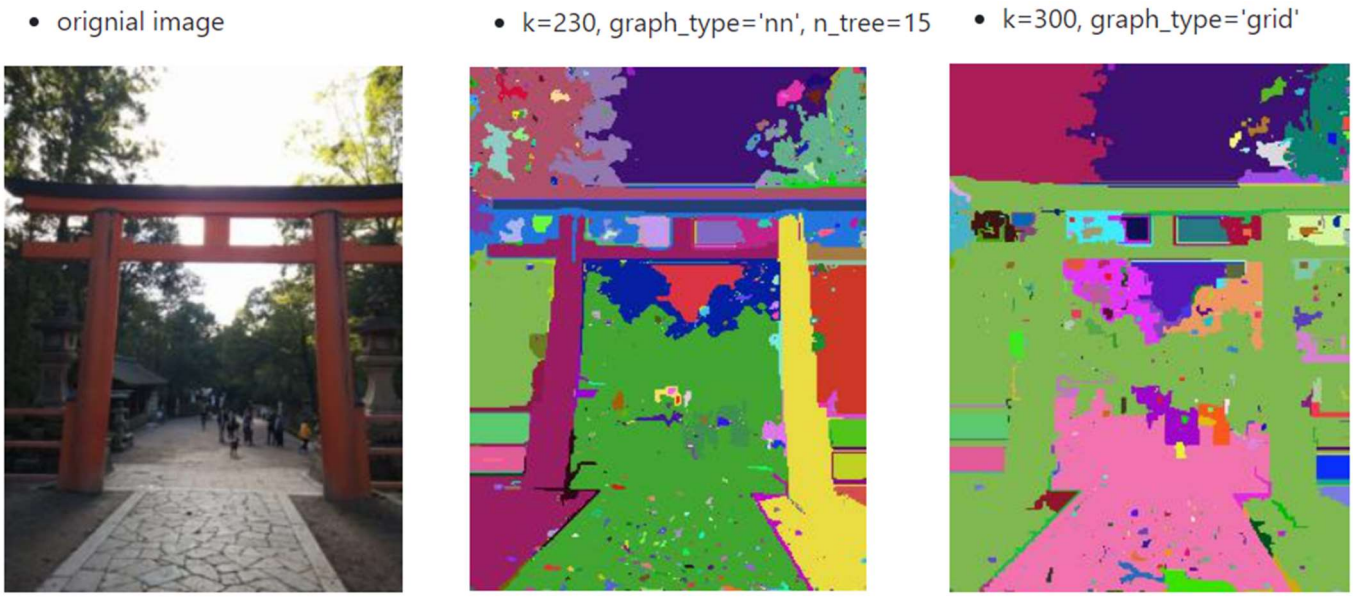

Figure 25. Sample images result

V. Discussions

After implementing and testing the various methods outlined above, our group decided to build our final software with a combination of methods 1 and 3 – stem detection and image color segmentation. These methods showed the most promising results. The end user can utilize either or both methods as a part of this system.

The first step when running our program is to pass in either a video or a folder of videos. Once the software determines what was passed to it, it creates a folder structure for the images and begins populating each folder with sampled frames from each video. While future iterations of this system could see things such as sampling rate and number of frames desired as parameters to the program, for now they must be changed within the sample_frames.py file itself. After the frames are sampled, we can move onto the next step. This will either encompass stem detection, image color segmentation (to pick out important objects), or both.

An efficient and dependable technique for segmenting tree stems in high-resolution aerial photographs is the tree stem identification algorithm proposed in this paper. The algorithm's ability to provide somewhat accurate results suggests that it has the potential to be used in applications for forest inventory and management. The technique is suitable for recognizing tree stems under various environmental situations because to its tolerance to noise and occlusion. The intricacy and density of the forest canopy, however, may have an impact on the algorithm's performance, which may restrict its application in some circumstances. The parameters of the program might be improved, and performance tests on various forest kinds and climatic circumstances could be conducted as part of future study. Overall, the findings imply that the suggested algorithm is a promising method for automatically detecting tree stems in aerial data.

For image color segmentation, we put each individual frame through the pipeline laid out in our section on method 3. The program then creates another set of folders, one for each video, that hold frames we flag to be interesting along with each of their clusters.

Method 2, WebODM, yielded some promising results. However, the extensive computation required for a single video led us to prefer the aforementioned methods. Additionally, there was a non-linear relationship between sampling rate and quality of the 3D reconstructed environment, so there was no clear answer as to how we could iteratively improve our results across time. Perhaps future research can look into building a more tunable tool with the WebODM API to build upon our work and solve some of these issues.

Method 4, "Monodepth2" depth detection, did not return good results, but holds a lot of promise for future research. Niantic Labs provides instructions on how any of the pretrained models they use can be fine-tuned for better results on specific datasets[1]. We conclude that this is the obvious next step. Pretrained models are revolutionizing the field of machine learning due to their power and generalizability. We can take an extremely complex model that has already been trained, retrain the final layer, and have a state-of-the-art model ready to be used for the specific purpose we fine-tuned it for.

After fine-tuning, future researchers could also look to the other pretrained models offered by Niantic Labs. Each model is listed on the Monodepth2 GitHub repository, and it would be trivial to implement each one (and perhaps fine-tune each one) to see which yields the best results.

Method 5, Graph based Image segmentation, did not yield good results but we believe that it could be fine tuned for better results. However, it might be difficult to achieve results comparable to the K means as a lot of images that this method was tested on were images with clear background and foreground. These situations might be difficult to achieve in a rainforest setting.

VI. Conclusions

Our team has built a tool we believe can be beneficial to those studying rainforest ecology in the field today as well as progressed various methods for future research to be done. While our WebODM 3D reconstruction and depth detection methods did not yield the results we were

hoping for, the next steps we laid out in their associated sections leave plenty to be done for those who decide to take on this problem in the future.

Our final program yields the promise of saving researchers valuable time in their endeavor to understand our world and the life around us, but it is not without its limitations. There are plenty of other factors that can be incorporated into our codebase in the future. We hope to see the passion around our field continue to progress as we see others build on what we have created in this project.

VII. Team Member Contributions

Alex – Methods 2, 3, and 4

Rucha – Method 1, Method 2 (separate implementation) and Method 5

VIII. Acknowledgements

We would like to thank Dr. Brooke for his guidance throughout the year and also for the resources for use (The computing laptop).

We would also like to thank our TA Aritra Ray for his help on brainstorming and mapping the course of our solution.

IX. Code Links

1. https://github.com/abzdel/rainforest_image_analysis
2. https://github.com/abzdel/monodepth_rainforest
3. https://github.com/RuePat/Efficient_Graph-based_Image_Segmentation
4. https://github.com/RuePat/PercepTreeV1
5. https://github.com/RuePat/PercepTreeV1/blob/main/Frames.py - Code to sample the frames from video

X. References

1. Norlab-Ulaval (no date) *Norlab-ULAVAL/Perceptreev1: Implementation of Grondin et al. 2022 'tree detection and diameter estimation based on Deep Learning'. also includes datasets and some of the pretrained models.*, *GitHub*. Available at: https://github.com/norlab-ulaval/PercepTreeV1 (Accessed: 03 May 2023).
2. https://developers.arcgis.com/python/guide/how-maskrcnn-works/#:~:text=Mask%20R%2DCNN%20architecture,label%20with%20a%20confidence%20score.
3. Godard, C., Mac Aodha, O., & Brostow, G.J. (2016). Unsupervised Monocular Depth Estimation with Left-Right Consistency. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6602-6611.
4. Ham, H., Wesley, J., & Hendra, H. (2019). Computer vision based 3D reconstruction: A review. International Journal of Electrical and Computer Engineering, 9(4), 2394.
5. Klose, F., Wang, O., Bazin, J. C., Magnor, M., & Sorkine-Hornung, A. (2015). Sampling based scene-space video processing. ACM Transactions on Graphics (TOG), 34(4), 1-11.
6. Kodinariya, T. M., & Makwana, P. R. (2013). Review on determining number of Cluster in K-Means Clustering. International Journal, 1(6), 90-95.
7. Luo, C., Yang, Z., Wang, P., Wang, Y., Xu, W., Nevatia, R., & Yuille, A.L. (2018). Every Pixel Counts ++: Joint Learning of Geometry and Motion with 3D Holistic Understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42, 2624-2641.
8. OpenDroneMap Authors ODM – A command line toolkit to generate maps, point clouds, 3D models and DEMs from drone, balloon or kite images.
9. Pell, T., Li, J. Y. Q., & Joyce, K. E. (2022). Demystifying the Differences between Structure-from-MotionSoftware Packages for Pre-Processing Drone Data. Drones, 6(1), 24. MDPI AG. Retrieved from http://dx.doi.org/10.3390/drones6010024
10. Ranjan, A., Jampani, V., Kim, K., Sun, D., Wulff, J., & Black, M.J. (2018). Adversarial Collaboration: Joint Unsupervised Learning of Depth, Camera Motion, Optical Flow and Motion Segmentation. ArXiv, abs/1805.09806.

11. Wang, C., Buenaposada, J.M., Zhu, R., & Lucey, S. (2017). Learning Depth from Monocular Videos Using Direct Methods. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022-2030.
12. Wang, Y., Chen, Q., & Zhang, B. (1999). Image enhancement based on equal area dualistic sub-image histogram equalization method. IEEE transactions on Consumer Electronics, 45(1), 68-75.
13. Yin, Z., & Shi, J. (2018). GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 1983-1992.
14. Zhan, H., Garg, R., Weerasekera, C.S., Li, K., Agarwal, H., & Reid, I.D. (2018). Unsupervised Learning of Monocular Depth Estimation and Visual Odometry with Deep Feature Reconstruction. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 340-349.
15. Zhou, T., Brown, M.A., Snavely, N., & Lowe, D.G. (2017). Unsupervised Learning of Depth and Ego-Motion