

Object Detection Final Report
December 15, 2017
Solomon Champion and Jean Fujikawa
Oahu Invasive Species Committee

Introduction

In 2017, the Oahu Invasive Species Committee (OISC) experimented with using various forms of digital imagery for mapping and detection of their target species. The imagery included video and images from UAV flights, Pictometry imagery, videos from fence line surveys, and Gigapan imagery. The accumulation of these and other remote sensing products over the years and the potential of accumulating more in the future, all of which would need review, led to OISC's exploration of computer-aided object detection.

Field associate Solomon Champion had an interest in learning to program in Python and applying it to object detection of OISC targets. He researched OpenCV and identified the object detection module as most appropriate for OISC's purposes. In conjunction with Operation Planner/Analyst, Jean Fujikawa, he explored the feasibility of using this open-source computer vision library to review our various imagery for our target species. The first phase of this study had the goal of exploring the feasibility of addressing the four following tasks:

1. Identify the target species in a single image.
2. Read a folder of images and produce a text file indicating the images with the target species identified.
3. Review a video and have the time stamp saved to a text file and save the image with the highlighted target as well.
4. Have the training selection be more documentable, visually to see what was selected for training the images.

The majority of this work was done using one of our target species, fountain grass *Cenchrus setacea*. We selected this species to create the test classifiers because fountain grass usually occurs in open canopy on cliffy terrain, thus minimizing the target being obscured from view. Also, there are thousands of images of fountain grass available for creating training images. Although miconia (*Miconia calvenscens*) is our primary target species, the vast majority of the miconia found on Oahu are immature plants and thus hidden in the understory. We do not have a lot of miconia on Oahu and consequently, we do not have a lot of miconia images.

Solomon researched the methods and code, and created the classifiers, while Jean provided overall project guidance and troubleshooting assistance with the code.

Methods

During researching of the OpenCV library various tutorials and example code have been reviewed and incorporated into the process. Much of the subsumed code, provided in the appendix, has been collated from work published online by Harrison Kinsley, and Adrian Rosebrock [2, 5]. A readymade repository for "*Creating a Cascade of Haar-Like Classifiers*" was available courtesy of Mahdi Rezaei of the University of Auckland [4].

With regard to the code there is only a few places it need be altered on case by case basis. Both the paths to the image(s) and/ or video files (JPG, BMP, MP4, etc.) and the cascade

classifier files (XML) should be explicitly denoted either at the command-line terminal, in script or by navigating the directory GUI (in the case of askdirectory.py).

The scaleFactor, minNeighbors and minSize are about the only parameters that need be altered occasionally. The Scale Factor parameter allows for the input image to be downsized. A scale factor value of 1.10 means that each time you downscale the input image by 10%, the value must be >1 to run the script. The Min Neighbors parameter specifying how many neighbors each candidate rectangle should have to retain it as a positive detection. A neighbor is merely how many neighboring detections are in the immediate vicinity of the detection window. Min Size is the pixel dimension of the detection window. The classification is a sliding window approach where the detection window is moved through the input image evaluating at each position the likelihood of a detection within.

Single image target identification

Single image target identification was accomplished by using the object detection module of the open-source package OpenCV. The module is based upon a HAAR cascade classifier [1]. When annotating it is important to keep in mind what features we choose for the HAAR Training and why we choose them. For example, the initial cascade classifier created for this study was that of a grass, *Cenchrus setacea* specifically. However, the best classifier for this grass came with imagery we annotated from the ImageNet synset for fountain grass. Notably, not all the “fountain grass” found in this synset was *C. setacea*. Some were closely related *Cenchrus* sp. some were species still in the genera designated *Pennisetum*. These distinctions are academic to our detection goals. By selectively annotating closely related bunch grasses we were able to train a cascade classifier that focused on the specific habit of these related bunch grasses rather than one bunch grass specifically. This may seem convoluted somewhat but in creating robust classifiers sample sets should be as large and distinct as possible. As a grass, fountain grass is inherently indistinct. This is a specific limitation to the detection of plants within an environment that do not have standard leaf shape but a distinct “fountaining” habit and a single growth point, as opposed to stoloniferous grasses.

Likewise, when annotating fountain grass it is also important to consider if the inflorescence will be included in the annotations. A persuasive argument for either inclusion or exclusion of the inflorescence might be made. If not included, we may be wasting a valuable feature for detection and if it is included, perhaps, an inadequately trained cascade would miss a juvenile plant. Likely having two cascade classifiers running independently of each other would be the solution to this *dilemma*.

Multiple image target identification

The functionality of being able to output which files contain detections is an extremely helpful feature. This is accomplished in the code by a call to the print function inside a “for” loop for the enumeration of each detection found in each file. Each detection is then output to a .txt file which is then counted up and output to another .txt file that is more readable, enumerating exactly how many detections were found in each image. This allows the user to prioritize the review process, given a robust classifier.

Arguably most useful for OISC detection purposes is the montage script, this program will batch detect in a directory of images. This script must be called from the command-line interpreter with the path to the cascade classifier set in the script. To run the script on a specific

directory of images say on the desktop, pass “python3 montage.py --images desktop/image_directory --sample 10”. Then Python should display a montage of the first 10 images in the said folder (image_directory), if the last part “--sample 10” is omitted it will default detect the first 100 images in the directory. Additionally, two text files will be output to the desktop. The first is output.txt is essentially a list of all the detections found in the directory, very verbose and repetitive. The second text file is named sort.txt and is essentially an account of all the detections in the first text file. Use this file to see which images have the most detections of the target species.

Video target identification

The ability to detect objects in video footage, along with object tracking, is in-built to the OpenCV library. Most of the video imagery trials were run on compressed .MP4 files from DJI Mavic UAV. For all intents and purposes the OpenCV library treats video files frame by frame as if it were a collection of still images.

Visual review of training images

There are a number of image annotator programs on the web, most output the object location in CSV format (comma separated value). For the purpose of HAAR Training the annotations should be a delimiter-separated value (DSV) text file padded with spaces, as opposed to commas or tabs (TSV). The annotator used on the OISC field computers is provided by Dr. Rezaei from the University of Auckland [4]. It works decently well although it does seem to log a number of key-stroke errors while performing the annotation of the number of objects in each image, this issue is thoroughly described in the documentation. For Mac OS, OpenCV provides an annotator natively that works very well without the key-stroke errors mentioned above. With both annotation software the size of the image being annotated cannot be too large without some alignment issues occurring with the cursor while performing annotations. Here we alter the dimensions of the imagery when needed, so as to annotate precisely however this invariably means some compression of the training set.

These annotators leave no readily visual documentation of what objects were annotated from the training set of images, apart from the output text file with pixel locations of the objects in a DSV format. This leaves much to be desired with regard to knowing which features of a given plant are best to annotate for training purposes. Likely, the DSV annotations could be used in conjunction with conserved image set to display the annotations post processing, however this will require a bit more research. Perhaps a better method would be for OISC to write its own image annotator that would output an image capture of the object being annotated, this however would of course create several hundred more redundant images.

Results

Single image target identification

The script titled askdirectory.py (Appendix A) is simply for selecting a single image from a directory detecting what target species is in the image, if any, according to the cascade classifier designated in the script. This script, when double-clicked, will open up a directory window that will allow the user to navigate to the desired image for detection. The path to the

cascade classifier should be set before detection is initiated and this variable path will need to be classifier, PENSET classifier, etc.). Once selected, the image with detections will be displayed and a JPEG of the detections created in the working directory.

Multiple image target identification

The montage.py program (Appendix B) is OISC's script for evaluating an entire directory of images simultaneously for detection of the target species, per cascade classifier designated in the script.

Video target identification

The script written for video is aptly named video.py (Appendix C). This is the one and only program we have for running the detection on video thus far. The code for video.py is in the appendix. This script similar runs with a simple double-click of the file. The paths to both the cascade classifier and the video file need to be explicitly outlined in the script prior to running the program.

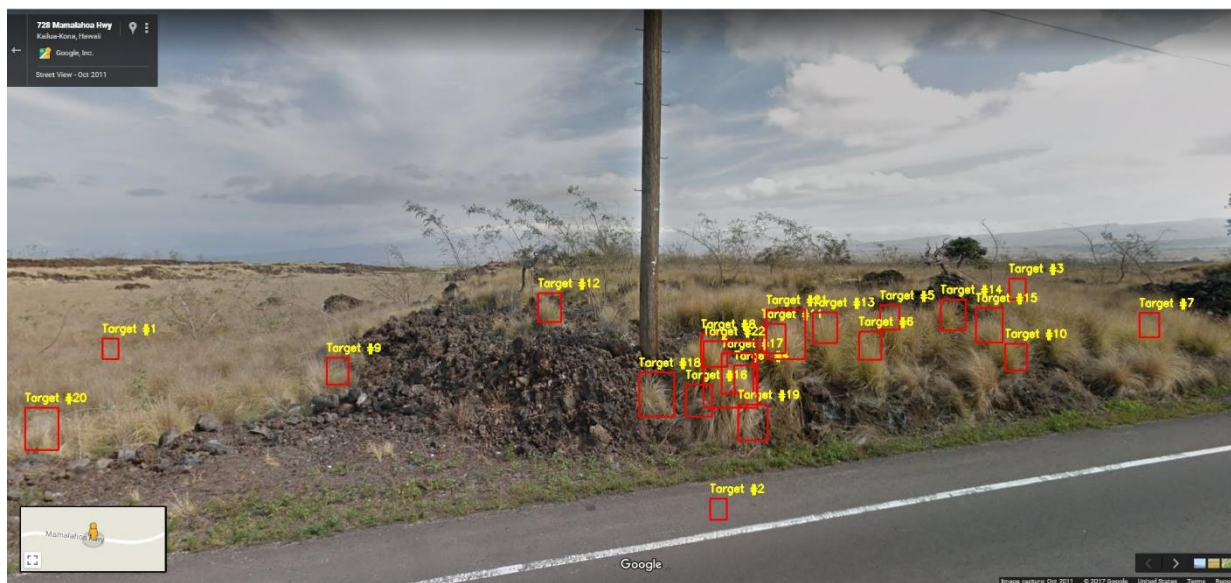


Figure 1. Screen shot of Hawaii Belt Road from Google Maps with an early iteration of the fountain grass classifier. Noticeably there is one false positive on the road likely this is an artifact of the annotation process as it is sometimes difficult to separate the rock from the grass when annotating this species.

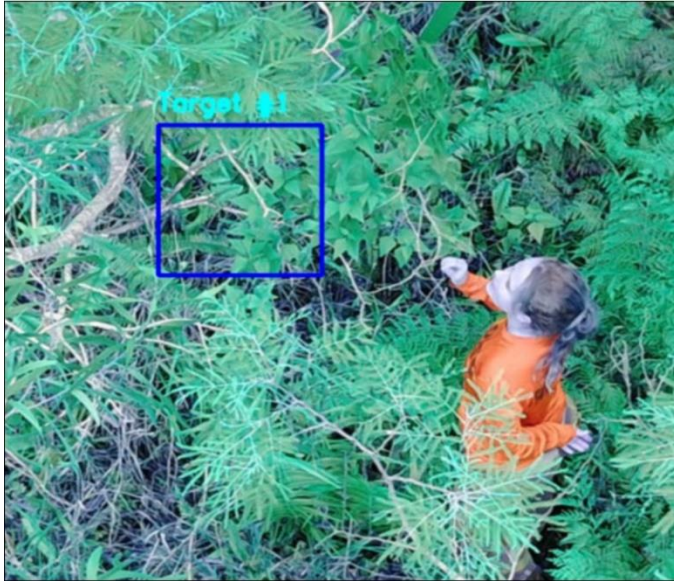


Figure 2. Screen capture from DJI Mavic video with detection bounding box output for *Chromolaena odorata*. Kahana Valley, Oahu. Photo: Derek Ford



Figure 3. *Miconia calvenscens* photographed from DJI Mavic Pro in Maunawili with bounding boxes displayed over detections, Oahu. Photo: Derek Ford

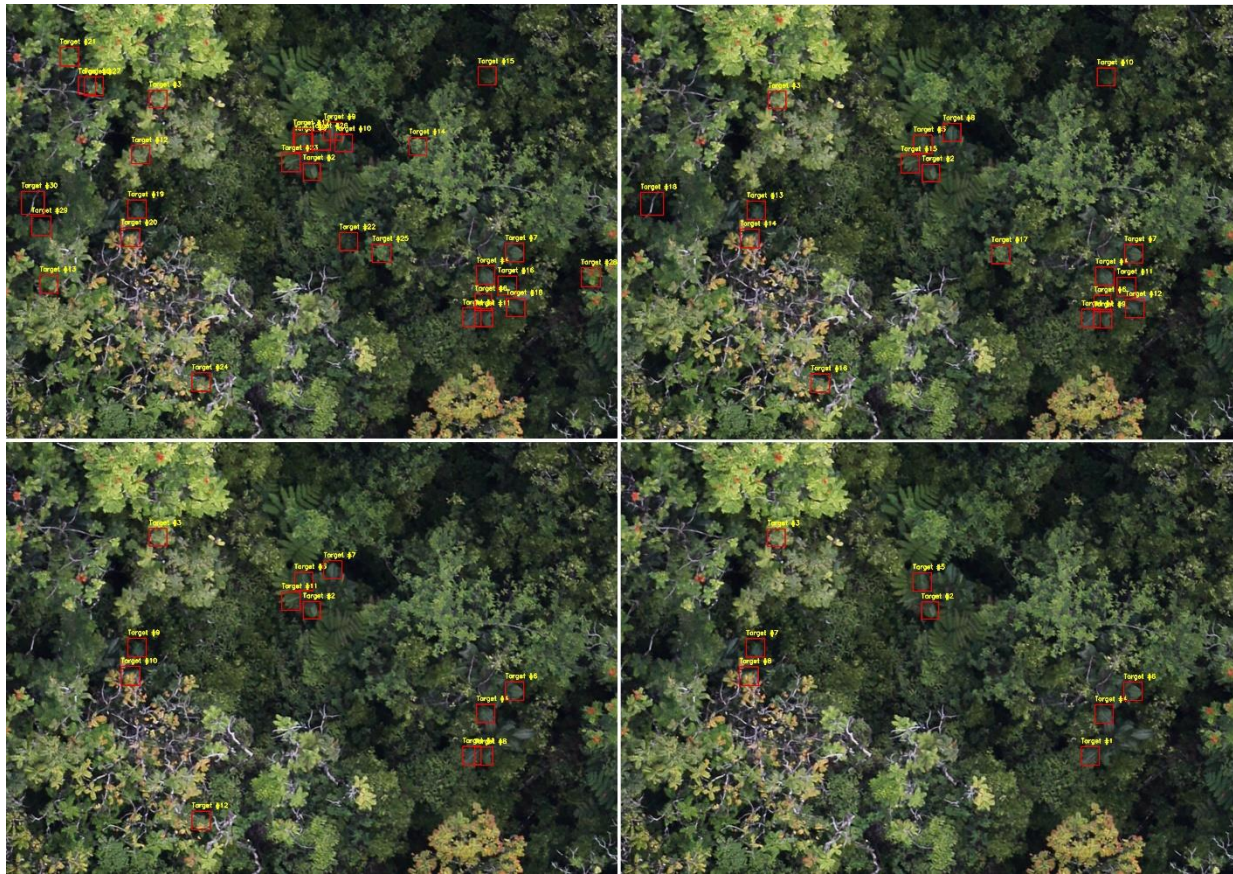


Figure 4. Resource Mapping Imagery from Maui *Miconia* Core Infestation with bounding boxes displayed around positive detections. Note the loss of false positives as same imagery repeated four times with the singular parameter of MinNeighbor changed, MinNeighbors = 1 & 2 on top , 3 & 4 on bottom respectively.

Discussion

Training

The original tutorial Solomon used to teach himself the process of creating HAAR classifiers can be found at: <https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/>. It is a very good tutorial and I would recommend it to anyone who would like to follow up this process. The original tutorial is quite thorough and differs from the tutorial provided by Dr. Mahdi Rezaei of the University of Auckland Computer Science Department in a number of ways. Firstly, perhaps the most obvious difference is this tutorial focuses on creating a classifier of a wristwatch, as opposed to faces. Secondly, the actual training of the cascade is done remotely on a Digital Ocean server. This is where Dr. Rezaei's method is more efficient, as it does not require us to rent a server. However, as our dataset increase in size and/or resolution it may become necessary to rent a server with superior processing power for the HAAR training purposes.

Image resolution

The process is applicable to OISC current strategy. Certain issues are impeding the

progress. There is a compression issue on both sides of the process. Training on compressed or otherwise lower resolution imagery is likely not ideal for the creation of high quality cascade classifiers. Likewise, the detection process stalls on high resolution images (large files) and detections creep frame by frame with HD footage. Better computers with more processing power and improved graphics card might ameliorate this. Additionally, a bit more elegant coding, “threading” specifically, may improve frame rate and make video detections a little more bearable to watch.

Our efforts thus far have been mostly focused on detecting targets from UAV imagery. However, most of our cascade classifiers have been trained on little to no UAV imagery this is probably the single greatest limitation of our study here at OISC. The cameras and pictures used in this study were not and could not be standardized for the HAAR training process, but they should be standardized, UAV photos for UAV classifier training. The OISC training image data sets have been a medley of images captured over years of OISC management of these targets species. During that time various cameras, file formats, and compression have been deposited in the OISC image database. Recently, we have been working to obtain more miconia imagery to build a better classifier from a UAV perspective. Some work has been done by Dr. Ryan Perroy’s lab at UH Hilo to assess canopy openness with UAV [3]. This work has amassed a decent collection of aerial imagery a mile out of Pahoa town on the Big Island, the kind of images perfect for training a UAV specific classifier. The Perroy lab has been kind enough to share these images with us and we intend to train a classifier specifically for UAV miconia surveys with the data.

Single image target identification

Cascade classifiers can be created a number of different ways [2, 4]. The way the cascade classifier works can be described as a decision tree where the features extracted from the training set are stacked with the most discriminating features at the top and more subtle features lower on the tree, thus as the detector moves through the classifier the features most unlikely to be the object of interest are rejected and those potentially positive features continue down the decision tree, thus “cascading” towards classification. Below is a description of the motivations for feature based classification rather than pixel-based from Viola-Jones, 2001.

“Our object detection procedure classifies images based on the value of simple features. There are many motivations for using features rather than the pixels directly. The most common reason is that features can act to encode ad-hoc domain knowledge that is difficult to learn using a finite quantity of training data. For this system there is also a second critical motivation for features: the feature based system operates much faster than a pixel-based system” [6].

It is important to note that if a less robust classifier is used then many false positives maybe detected. The method for creating the best possible classifier is at best conceptual and dependent upon the target species morphology, phenology, habitat, etc. For example, *Miconia* has very large leaves with prominent venation and a deep purple under-leaf. However, at least on Oahu, *Miconia* is often obscured by canopy and HAAR cascades classifiers are not dependent on color, but upon features derived from pixel intensity (*i.e.*; greyscale). These parameters should inform our image annotations for the purpose of HAAR training. Meaning leaf color, adaxial or abaxial will be of very little use, additionally we can surmise most plants will be at least somewhat obscured by surrounding vegetation. Perhaps then the best feature we can ask for is a single leaf poking above the canopy in view of camera.

The better classifier created, the more reliable the detections will be. Ideally, the classifier

would result no false positives and no false negatives. As of this report there are three methods for object detection.

Multiple image target identification

The arguably most useful for OISC detection purposes is the montage script, this program will batch detect in a directory of images. Additionally, two text files will be output to the desktop. The sort.txt is essentially an account of all the detections in the first text file. Use this file to easily see which images have the most detections of the target species.

Video target identification

High definition videography is increasingly possible with the advent of UAV technology. The DJI Mavic is capable of shooting at 4K 30 fps, however for OISC's purposes flights are usually recorded at 1080p resolution (D. Ford, pers. comm.). The OISC field computers are HP and Dell desktop with i3 and i7 processors, respectively. Each running Windows 10 operating systems. Additionally, the program has also been run successfully on a MacOS High Sierra, operating with an i5 processor. On all the aforementioned platforms, video files shot at 1080p will run extremely slowly while performing object detection. It has therefore been necessary to compress all of the imagery used in the program significantly, ~ 50% compression. This has usually been accomplished with the ImageMagick and FFMPEG packages for pictures and video, respectively. This compression likely has a significant effect on detection rate, although we have made no attempt to quantify this in this study.

These restrictions in combination with DJI's limited accessibility to their video encoding process have stymied progress in producing as useful a program for video as we have created for simple images. Efforts to retrieve a time stamp from detections in the video files should continue but ultimately DJI may not allow access to the metadata associated with its video files. A window capture of each individual detection may be an achievable output with the current design. This will require a little bit more research and will still require someone to review the images captured.

Visual review of training images

Writing the code for a better image annotation program to suit OISC's need for documenting the images going into the HAAR Training is a good future goal. There is almost certainly a way to run various cascades classifiers concurrently, thus doing away with path editing the classifier when moving from one target species to the next. This will require a bit more research.

Future research

This study so far has been a short foray into creating robust cascade classifiers. What would benefit the process now would be any effort that attempted to replicate the professional grade cascade classifiers. Most tutorials recommend somewhere between a 1000-5000 positive imagery annotations and twice as many negative images. The most robust classifier we have trained in this study for comparison is with 748 positive (*Miconia* leaves) annotations with 1571 negative images. While this classifier (miccal-cascades-30stages.xml) is perhaps under sampled

by most recommendations it is rather good at detecting *Miconia calvescens*, this is encouraging to say the least. 748 positive annotations takes a few hours for one person to do, the HAAR training can easily take upwards of 12 hours on a slower machine, but does not need to be supervised once begun. While 5000 positive annotations sounds like an overwhelming number we can define as many positive annotations in an image as there are visible leaves (in the case of *Miconia*) and most UAV mapping flights will be largely negative imagery of the variety perfect for training a robust classifier.

At the base of all these processes is creating a more robust cascade classifier consisting of over 1,000 positive images and twice as many negative images. The ideally the positive images would be specific to a particular target species and image collection method (*e.g.* Gigapan, UAV). As mentioned in the discussion the Perroy lab has shared the images from their work on the Big Island with assessing canopy openness and miconia detection. This data should be used in the next step in creating a robust miconia classifier. As for fountain grass and devil weed, it is likely OISC will need to collect more UAV imagery for themselves on these species to create a robust classifier. The data from Big Island would likely prove useful to Oahu's efforts as well as only the annotations of the target species would be used in the training and the negative directory of images could contain a large sample of typical Oahu vegetation to help parse out the target plants from the surrounding forest.

A way of quantifying detection rate is still needed to assess how well a classifier works. A metric for the number of positive detections versus false positive, coupled with a similar comparison of positive detection and false negatives. This would allow for statistical analysis of how well the cascade classifiers are trained for the target species.

For video target identification, better computational hardware, graphics card, processor, and monitors will help with the review process.

As for accessing metadata (time stamps, GPS coordinates) in the DJI footage, perhaps there is a way to get it but it may invalidate the licensing. Maybe another brand of UAV or one built specifically for the purpose of remote sensing would allow for access to these information or maybe displaying the timestamp at the time of recording is possible so it is saved directly on the footage.

Conclusion

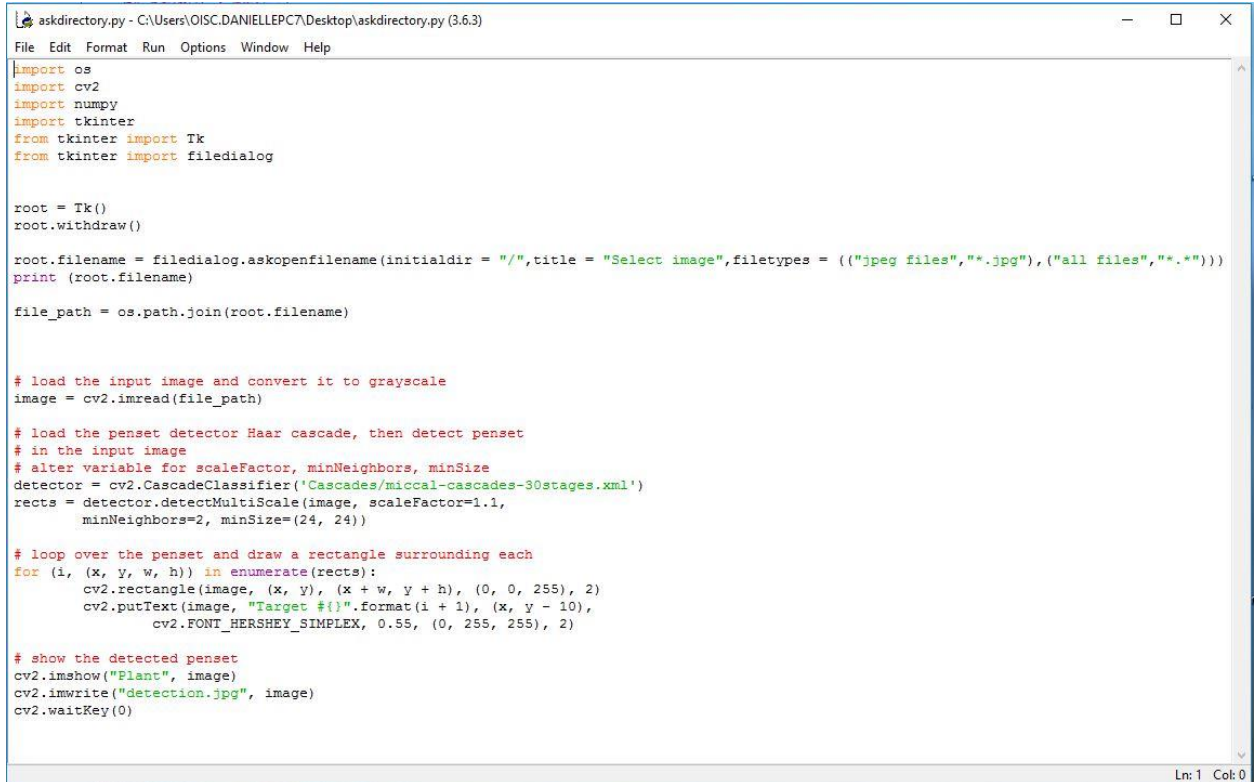
This study has helped to create a feasible avenue for reviewing the massive amount of image data that can be collected with remote sensing technologies. The current versions of the programs written are functional, however, there is certainly much room to optimize the code written here and to automate the video review. Future work would include accumulating at least 1,000 positive annotations per target species and image collection method of interest. These images would then be used to create better cascade classifiers to use as the basis of detecting invasive species in single large images, a collection of images, or in video.

As part of this study, initial cascades were created for fountain grass, devil weed, and miconia. In the future, cameras and pictures should be standardized for the HAAR training process. Only with better HAAR cascades can we hope to eliminate false negatives and reduce false positives to an acceptable level.

References

- [1] “Cascade Classification.” Cascade Classification — OpenCV 2.4.13.4 Documentation, docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html.
- [2] “Creating Your Own Haar Cascade OpenCV Python Tutorial.” Python Programming Tutorials, pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/.
- [3] Perroy, Ryan L., et al. “Assessing the Impacts of Canopy Openness and Flight Parameters on Detecting a Sub-Canopy Tropical Invasive Plant Using a Small Unmanned Aerial System.” ISPRS Journal of Photogrammetry and Remote Sensing, vol. 125, 2017, pp. 174–183., doi:10.1016/j.isprsjprs.2017.01.018.
- [4] Rezaei, Mahdi. “Creating a Cascade of Haar-Like Classifiers: Step by Step.” *Creating a Cascade of Haar-Like Classifiers: Step by Step*, University of Auckland.
- [5] Rosebrock, Adrian. “Be Awesome at OpenCV, Python, Deep Learning, and Computer Vision.” PyImageSearch, www.pyimagesearch.com/.
- [6] Viola, P., and M. Jones. “Rapid Object Detection Using a Boosted Cascade of Simple Features.” Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, doi:10.1109/cvpr.2001.990517.

Appendix A – askdirectory.py



```
askdirectory.py - C:\Users\OISC.DANIELLEPC7\Desktop\askdirectory.py (3.6.3)
File Edit Format Run Options Window Help

import os
import cv2
import numpy
import tkinter
from tkinter import Tk
from tkinter import filedialog

root = Tk()
root.withdraw()

root.filename = filedialog.askopenfilename(initialdir = "/",title = "Select image",filetypes = (("jpeg files","*.jpg"),("all files","*.*")))
print (root.filename)

file_path = os.path.join(root.filename)

# load the input image and convert it to grayscale
image = cv2.imread(file_path)

# load the penset detector Haar cascade, then detect penset
# in the input image
# alter variable for scaleFactor, minNeighbors, minSize
detector = cv2.CascadeClassifier('Cascades/miccal-cascades-30stages.xml')
rects = detector.detectMultiScale(image, scaleFactor=1.1,
    minNeighbors=2, minSize=(24, 24))

# loop over the penset and draw a rectangle surrounding each
for (i, (x, y, w, h)) in enumerate(rects):
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.putText(image, "Target #{0}".format(i + 1), (x, y - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255), 2)

# show the detected penset
cv2.imshow("Plant", image)
cv2.imwrite("detection.jpg", image)
cv2.waitKey(0)
```

Ln: 1 Col: 0

Appendix B – montage.py

```
montage.py - C:\Users\OISC.DANIELLEPC7\Desktop\montage.py (3.6.3)*
File Edit Format Run Options Window Help

# USAGE
# python3 montage.py --images images --sample 10
import argparse
import numpy as np
import cv2
import collections
import os
from imutils import build_montages
from imutils import paths

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--cascade",
                default="penset-cascade-25stages.xml",
                help="path to plant haar cascade")
ap.add_argument("-i", "--images", required=True,
                help="path to input directory of images")
ap.add_argument("-s", "--sample", type=int, default=100,
                help="# of images to sample")
args = vars(ap.parse_args())

# grab the paths to the images,
imagePaths = list(paths.list_images(args["images"]))
imagePaths = imagePaths[:args["sample"]]

# initialize the list of images
images = []

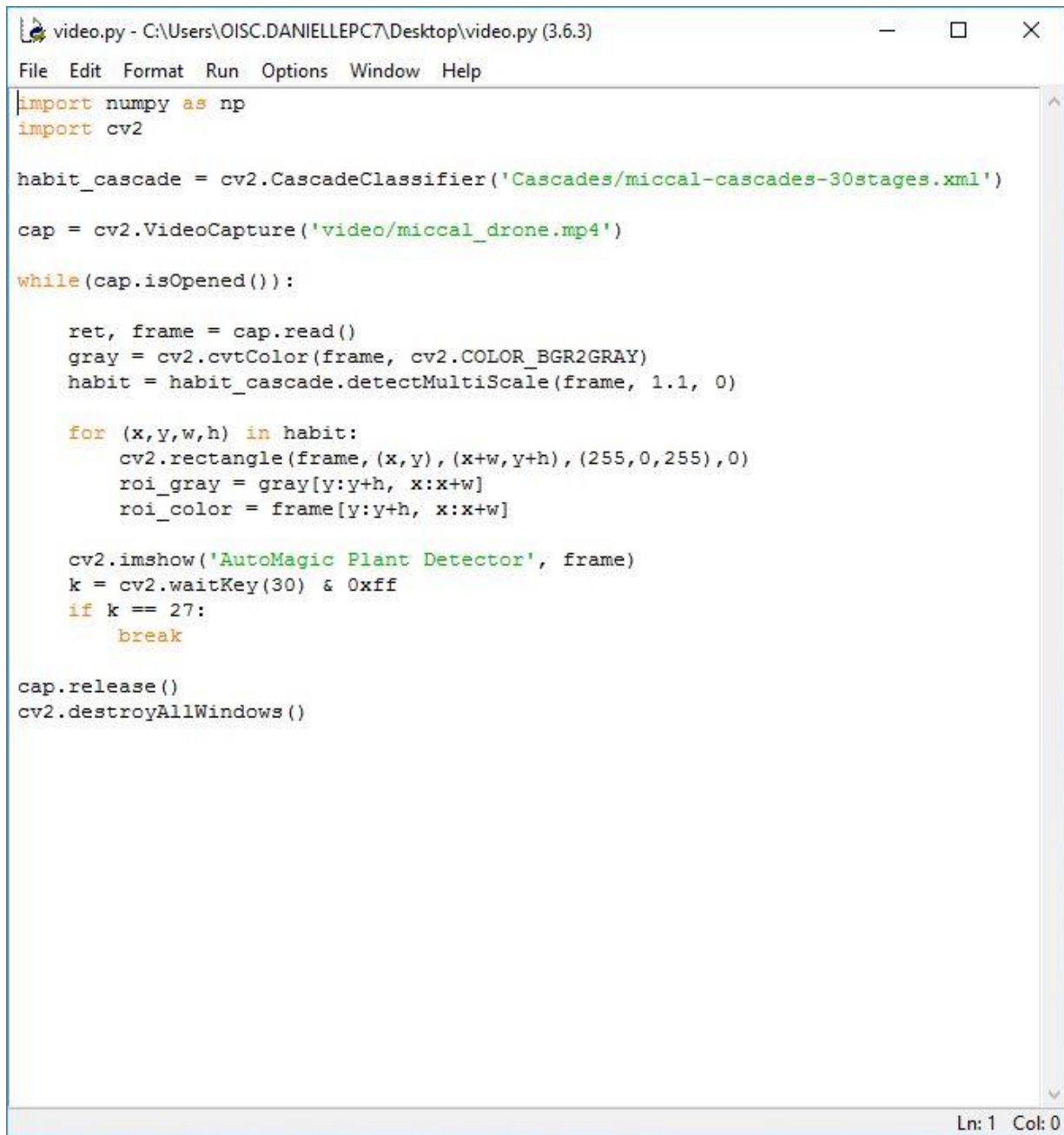
for imagePath in imagePaths:
    image = cv2.imread(imagePath)

# load the penset detector Haar cascade, then detect penset
# in the input image
# alter variable for scaleFactor, minNeighbors, minSize
# update the list of image
    detector = cv2.CascadeClassifier(args["cascade"])
    rects = detector.detectMultiScale(image, scaleFactor=1.01, minNeighbors=1, minSize=(24, 24))
# loop over the penset and draw a rectangle surrounding each
    for (i, (x, y, w, h)) in enumerate(rects):
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
        cv2.putText(image, "Target #{}".format(i + 1), (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 255, 255), 2)
        print(imagePath, file=open("output.txt", "a"))

    # construct the montages for the images
    montages = build_montages(images, (200, 300), (10, 10))
    images.append(image)
# loop over the montages and display each of them
for montage in montages:
    cv2.imshow("Results", montage)

# Put waitKey in for loop for image by image and press any key to exit montage and
# Create sorted output file "image with image name and number of detections"
cv2.waitKey(0)
with open('output.txt') as infile:
    counts = collections.Counter(l.strip() for l in infile)
for line, count in counts.most_common():
    print(line, count, file=open("sort.txt", "a"))
```

Appendix C – video.py



```
video.py - C:\Users\OISC.DANIELLEPC7\Desktop\video.py (3.6.3)
File Edit Format Run Options Window Help

import numpy as np
import cv2

habit_cascade = cv2.CascadeClassifier('Cascades/miccal-cascades-30stages.xml')

cap = cv2.VideoCapture('video/miccal_drone.mp4')

while(cap.isOpened()):

    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    habit = habit_cascade.detectMultiScale(frame, 1.1, 0)

    for (x,y,w,h) in habit:
        cv2.rectangle(frame, (x,y), (x+w,y+h), (255,0,255), 0)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]

    cv2.imshow('AutoMagic Plant Detector', frame)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Ln: 1 Col: 0

Appendix I

Installing OpenCV for the Purpose of Remote Sensing

OpenCV is an Open Source Computer Vision Library with a number of different modules for gaining high-level understanding of digital imagery (*i.e.*; photographs, and videos). Essentially automating several tasks that would otherwise have to be accomplished by the human eye. For the purpose of detecting plants in UAV, Gigapan or satellite imagery, the object detection module is discussed here.

Described below is the installation of the OpenCV library with Python bindings and its associated packages (OpenCV is originally designed for the C/C++ so the Python bindings are necessary for the scripts in the appendix, <https://www.opencv.org>). Here we describe the preferred Python 3 installation, however 2.7 is also available, for Windows 10 OS.

Required Packages

NumPy - <http://www.numpy.org>

Python 3 - <https://www.python.org/downloads/>

OpenCV Python Bindings - <https://pypi.python.org/pypi/opencv-python>

Optional

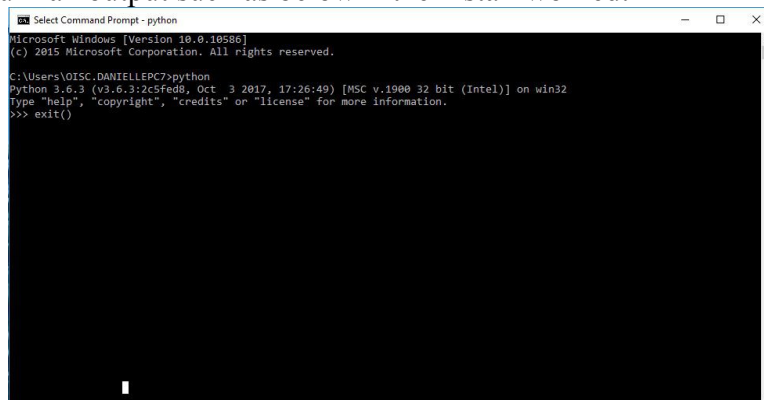
ArgParse - <https://pypi.python.org/pypi/argparse>

Matplotlib - <https://matplotlib.org/users/installing.html>

PIP - <https://pypi.python.org/pypi/pip>

INSTALLATION (PIP)

1. Go to Python.org click on downloads tab and choose current version of Python 3.X.X
2. Open downloaded python-3.X.X.exe file and run standard installation complete with PIP, IDLE, and Documentation. Check the box to install to PATH.
3. With the command-line interpreter open type “python” and hit enter. You should see the interpreter return an output such as below if the install worked:

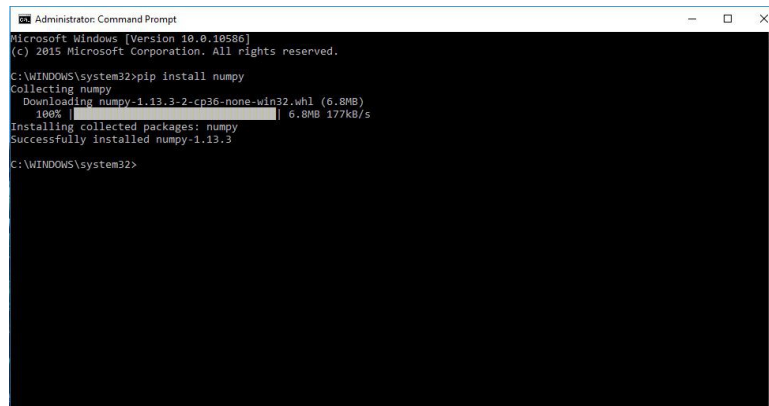


```
Select Command Prompt - python
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\OISC.DANIELLEPC7>python
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> exit()
```

4. Open up the command-line interpreter as an Administrator. This can be done by pulling up the windows search bar (windows key + S on keyboard) typing “Command Prompt” or CMD, right clicking on the app and selecting run as administrator.

5. To install NumPy, type “exit()” to leave the python interpreter and hit enter. Next type “pip install numpy” and hit enter. You should see the interpreter return an output such as:

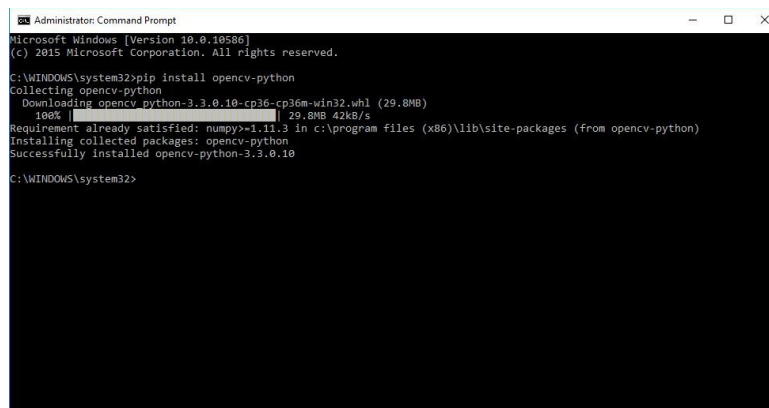


```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip install numpy
Collecting numpy
  Downloading numpy-1.13.3-2-cp36-none-win32.whl (6.8MB)
    100% |#####| 6.8MB 177KB/s
Installing collected packages: numpy
Successfully installed numpy-1.13.3

C:\WINDOWS\system32>
```

6. To install OpenCV with Python Bindings, you may type into the same command-line interpreter “pip install opencv-python” and hit enter. You should see the interpreter return an output like:



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip install opencv-python
Collecting opencv-python
  Downloading opencv-python-3.3.0.10-cp36-cp36m-win32.whl (29.8MB)
    100% |#####| 29.8MB 42kB/s
Requirement already satisfied: numpy>=1.11.3 in c:\program files (x86)\lib\site-packages (from opencv-python)
Installing collected packages: opencv-python
Successfully installed opencv-python-3.3.0.10

C:\WINDOWS\system32>
```

7. Check everything installed OK, to do this in the command prompt call the python interpreter by typing “python” hit enter. You should see a similar output to step 3. Next type “import cv2” hit enter. If the line passes without errors output in the command-line interpreter then OpenCV package is installed and loaded. Next type “import numpy” to check it installed, likewise if the line passes without error NumPy is installed and loaded. Congratulations! You've now installed all the requisite packages to run OpenCV.