Astro Pi Proposal – Stage 1 – Life On Earth
13-Oct-2021

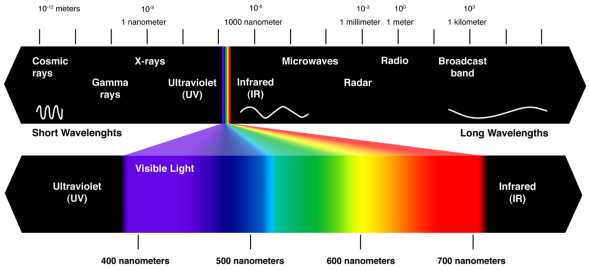
Using NDVI Analysis to Calculate Land Area, Plant Coverage & Health to Predict Vegetation Change Using Previous Historical Imaging – While Displaying the Data Gathered in a Highly Interactive UI

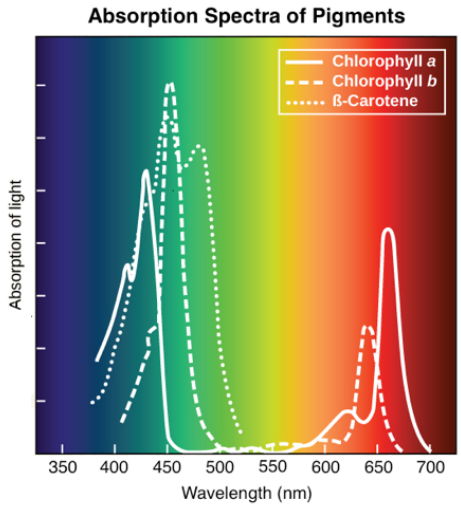
In our planned experiment we exploit the Raspberry Pi's Near Infrared "Noir" Camera alongside the Normalized Difference Vegetation Index ($NDVI$) to quantify current vegetation and compare it with historical data. Our aim is to combine the two datasets to train an auto-regressive model that can predict future changes in both plant coverage and health. We will take images in set intervals so when plotted on our interactive UI we can properly communicate how the measured health, coverage etc.. varies with location.

What is NDVI?

Green plants contain a chemical called chlorophyl stored in the chloroplasts in plants which helps to use the light energy from the sun into carbon dioxide & water. This process is called photosynthesis. Chlorophyll cannot absorb 100% of the sun's light, in the EM spectrum there are forms of light invisible to the naked human eye.

- UV is invisible to humans and contributes to sunburn
- Infrared allows us to feel heat i.e. when you put your hand near a fire
- Others include: microwaves, gamma, x-rays etc...





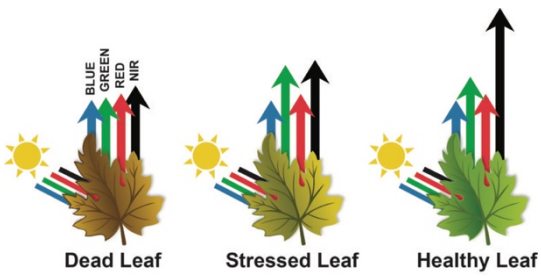
We can also see how chlorophyll only absorbs some of the suns light in photosynthesis, the graph below demonstrates this. Notice how the green light is not absorbed and hence the color of chlorophyll.

Likewise, these "green plants" (not including plants like Acers which have reddish leaves) don't handle infrared radiation (heat) well and aim to reflect as much as possible.

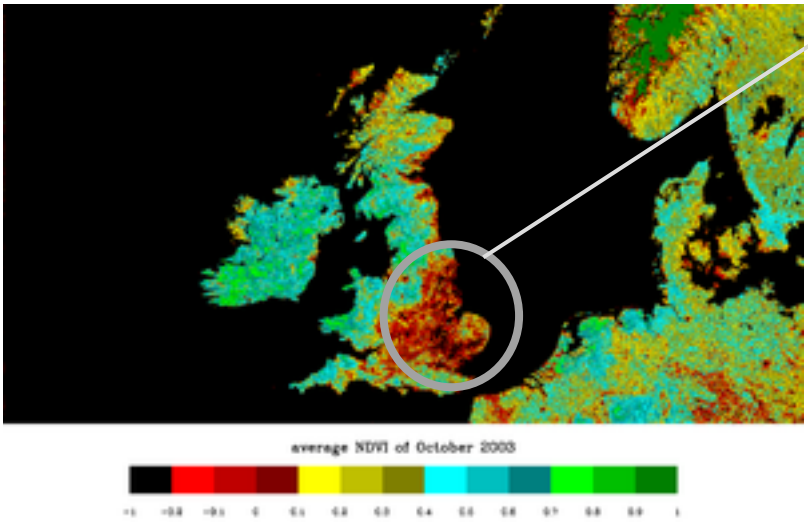
A contemporary 12 mega-pixel camera can detect many types of light including IR. Since this doesn't match what the human eye sees (visible light) a filter is added to the camera that this radiation is absorbed and doesn't reach the cameras filter.

From this we can comfortably conclude **healthy plants reflect lots of IR whereas dying plants absorb lots of IR.** If you were to remove the IR filter from a light you would see a blue-green colour – this is what we can utilise to measure plant health / coverage.

We propose to use an infrared filter to detect amounts of infrared light that is reflected therefore measuring the health of the plant.



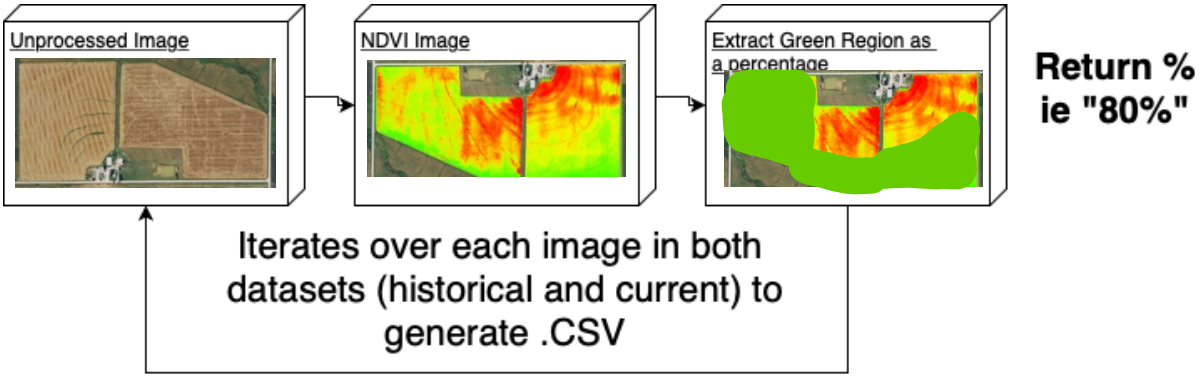
Example NDVI of UK



You can clearly see in a NDVI image that an urban area has less vegetation, this is the quality of imaging that we hope to get back from the images taken on the ISS and from the existing historical data. Also note how this was taken in Autumn so vegetation was already lacking which helps to better communicate a lack of vegetation in this area.

Autoregressive Model – Predictions

Note: since we would be using a regressive algorithm over a classification algorithm the data needs to be numerical double values. These would be calculated by taking a NDVI image → extracting number of green pixels → calculating this as a percentage of the whole image.



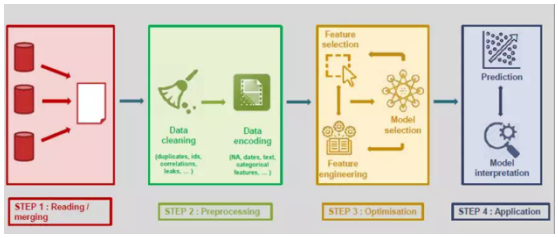
We could employ a library called $PyCaret$ to train the model using default hyperparameters to ensure minimum parameter tuning later on in the training process. We should evaluate the performance metrics using cross-validation returning the trained model object. For the regression we can use: $MAE, MSE, RMSE, R2, RMSLE, MAPE$ (Scikit Learn) algorithms since they have already been tested and we train the model on all of them and select the model based upon the highest accuracy. We can also blend the models together to boost accuracy using a boosting modifier ie. catboost regressor.

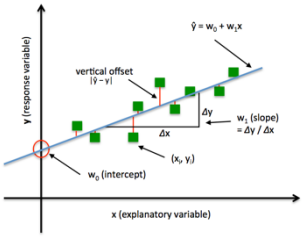
The models below have been taken from the Scikit learn libraries API and these are the models we would compare the accuracy of

Abbreviation	Full Name	Abbreviation	Full Name
LR	Linear Regression	Lasso	Lasso Regression
Ridge	Ridge Regression	En	Elastic Net
LAR	Least Angle Regression	OMN	Orthagonal Matching Pursuit

BR	Bayesian Ridge	ARD	Automatic Relevance Determination
PAR	Passive Aggressive Regressor	Ransac	Random Sample Consensus
... 15 more	...	...	...
Huber	Huber Regressor	Kr	Kernel Ridge
SYM	Support Vector Regression	KNN	K Neighbors Regressor

The diagram shows the steps involved in gathering the data where reading/merging is the combination of the historical and measured data, pre-processing is cleaning up the CSV file, optimisation is the model building (as discussed above) and the application is the prediction in the GUI as explored below.



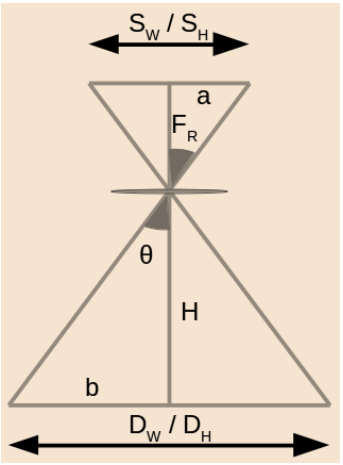


To better explain how the regression model works, I often refer to this diagram which is true for all cases of linear regression. The green squares represent the datapoints and the independent variable is on the X axis which is would be the number of measured "healthy" pixels (land area) which would then be used as our predictor – this is what we can imagine but for a larger number of datapoints.

Using a Numerical Method to Calculate Land Coverage on a "Per pixel basis".
We would estimate the area of a single pixel using a "ground sampling surface" method which would give a numerical float proportionate to the amount of land in the given frame. By employing this method we can ensure that the results won't be skewed if (for example) 30% of the screen was covered in non-vegetation aka. the sea by using edge-detection on the image using CV2 (python) beforehand.

Stage 1 - Edge Detection:
"Edges are characterized by sudden changes in pixel intensity. To detect edges, we need to go looking for such changes in the neighbouring pixels." By exploiting Sobel Edge Detection alongside Canny Edge Detection we can find the edges (contours) of a land mass to scale our initial value by. After reading the image we'll blur it using Gaussian Blur this reduces the overall noise in the image so less computation is used when jagged edges don't represent the predominant edges that we are looking for.

Stage 2 – Ground Sampling Surface – inspired method used in a different scientific publication



Measurements needed:

- SW (sensor width)
- FR (Focal length)
- H (ISS height)
- DW(distance covered in width direction)
- SH (sensor height)
- FR(Focal length)
- DH(distance covered in height direction)
- IW (image width

$$\tan(\theta) = \frac{a}{F_R} = \frac{b}{H} \rightarrow \frac{2 \cdot a}{F_R} = \frac{2 \cdot b}{H} \rightarrow \frac{S_W}{F_R} = \frac{D_W}{H} \rightarrow \rightarrow \rightarrow D_W = \frac{H \cdot S_W}{F_R}$$

This method use the tan law to take the inverse angle of the sensor which is then projected onto the face of the camera. This is demonstrated in the diagram above since a is inverted into b and S_W/S_H is translated into D_W/D_H. This gives us an accurate representation of a single pixel and what it represents in real life. 30,000m^2 would be a sensible representation of a single-pixel from space and this would be used in the later calculations.

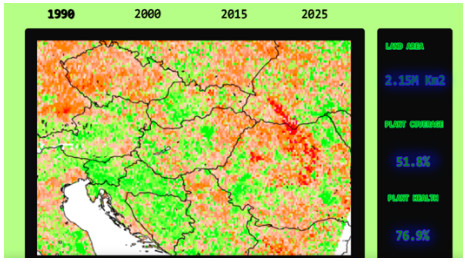
Stage 3 – Combining the edge detection and Ground Sampling Surface
Given the real area a single pixel covers. "aI" (Actual Image) is the iW (image Width) x iH (image Height) divided by the number of green pixels (gP) multiplied by the area of a single pixel (3000 in example) and divided by 100 to gain percentage form which is the single numerical value used in the predictive model.

$$aI = \frac{\frac{iW \cdot iH}{gP} \cdot 3000}{100}\%$$

How we could improve further – 3rd Party Feedback
Although out of scope of the experiment, the observation was made that the algorithm wouldn't detect the biodiversity - one of the largest problems isn't deforestation but what is being replaced. Ie. in the Amazon jungle the flora/fauna is being replaced with palm oil trees which isn't better for the environment since that biodiversity is wiped out. In response, a different experiment with a much higher resolution camera could use object detection to look at wildlife patterns. This feedback highlights the importance of tackling all aspects of the problem ie. our edge detection model since the data on its own can be misleading without looking at the context around the data which is both applicable in the example highlighted and our wider experiment.

Prototype of the Interactive UI
We noticed that one of the largest issues with current experiments was a difficulty interpreting the raw data. To solve this we hope to create a GUI that shows the collected data (both historical and prediction) with various statistics in a friendly and manageable UI. We have prototyped this in python (pygame) and provide a video / screenshots alongside the project demonstrating how it works.

The green line representing the flight path and the yellow dots show the datapoints collected which when clicked expand into a detailed view:



In this GUI you have access to the land area, plant coverage & plant health and as you click each year the image of the NDVI which changes and 2025 shows the predicted statistics. Remembering that this is a prototype and more features should be added.

You can see a video walkthrough here: https://www.youtube.com/watch?v=pVnqQlVK6Ik

Prototyping the NDVI analysis – from Writeup To Minimum Reproducible Code.

We can take this from writeup into program form using python – the code here has been adjusted from an existing Github Repository (https://github.com/robintw/RPiNDVI).

```
1.  import sys
2.  import time
3.  import numpy as np
4.  import cv2
5.  import picamera
6.  import picamera.array
7.
8.
9.  """
10. Formula Used for NDVI:
11.                 NDVI = (NIR − Red)/(NIR + Red)
12.
13. """
14.
15.   def label(image, text):
16.       """
17.       Labels the given image with the given text
18.       """
19.       return cv2.putText(image, text, (0, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, 255)
20.
21.
22.
23.
24.   def contrast_stretch(im):
25.       """
26.       Scales the contrast of the image
27.       """
28.       in_min = np.percentile(im, 5)
29.       in_max = np.percentile(im, 100)
30.
31.
32.       out_min = 0.0
33.       out_max = 255.0
34.
35.
36.       out = im - in_min
37.       out *= ((out_min - out_max) / (in_min - in_max))
38.       out += in_min
39.
40.
41.       return out
42.
43.
44.
45.
46.   def run():
47.       with picamera.PiCamera() as camera:
48.           # Set the camera resolution
49.           x = 400
50.           camera.resolution = (int(1.33 * x), x)
51.           # Various optional camera settings below:
52.           # camera.framerate = 5
53.           # camera.awb_mode = 'off'
54.           # camera.awb_gains = (0.5, 0.5)
55.           camera.hflip = True
56.           camera.vflip = True
57.           if len(sys.argv) == 2:
58.               camera.rotation = sys.argv[1]
59.
```

```
60.
61.          # Need to sleep to give the camera time to get set up properly
62.          time.sleep(1)
63.
64.
65.          with picamera.array.PiRGBArray(camera) as stream:
66.              # Loop constantly
67.              while True:
68.                  # BGR NOT RGB!!!
69.                  camera.capture(stream, format='bgr', use_video_port=True)
70.                  image = stream.array
71.
72.
73.                  # Get the individual colour components of the image
74.                  b, g, r = cv2.split(image)
75.
76.
77.                  # NDVI Calculations
78.                  bottom = (r.astype(float) + b.astype(float))
79.                  bottom[bottom == 0] = 0.00001  # Make sure we don't divide by zero!
80.
81.
82.                  ndvi = (r.astype(float) - b) / bottom
83.                  ndvi = contrast_stretch(ndvi)
84.                  ndvi = ndvi.astype(np.uint8)
85.
86.
87.                  # Annotations
88.                  label(image, 'original')
89.                  label(ndvi, 'NDVI')
90.                  cv2.imshow('NDVI Image', ndvi)
91.                  cv2.imshow('Original Image', image)
92.
93.
94.                  stream.truncate(0)
95.
96.                  c = cv2.waitKey(7) % 0x100
97.                  if c == 27:
98.                      break
99.      cv2.destroyAllWindows()
100.
101.
102.          if __name__ == '__main__':
103.              run()
104.
```

Works Cited

https://projects.raspberrypi.org/en/projects/astropi-ndvi/1 - Capture plant health with NDVI and the Raspberry Pi

https://learnopencv.com/edge-detection-using-opencv/ - Open CV – edge detection