Please refer to the code in the following GitHub page if you need it:
https://github.com/vcovasan/mars_rover_vic/blob/master/object_bounder_video.py
You will need to be invited to the GitHub project first.

1.1 Statement of the problem

To make the following OpenCV based technique for the mars rover, we made the following assumptions

Problem statement: Detect (or localise) this green object, and output a YES or NO as to whether this green object is seen .

Assumptions: The object is always green, and is in the sand. There won't be any other 'green'-ish objects around. There is probably only one object in the field of view of the rover.

1.2 A simple ballpark technique to solve the problem.

As the problem is well-defined here, we can use a very simple OpenCV Color Range function to detect the presence of any 'variant' of green. This should work well to localise the object. The object's presence can be confirmed by the presence of a green object

Software/language used: **Python 3 and OpenCV version 4**

1.3 Head-first approach into detecting object by its color

- Apply Blur (Gaussian Blur) with kernel (5,5) to the picture. Why (5,5)? Well, it doesn't blur the picture to the point where it's barely recognisable - feel free to adjust this. This is a standard procedure in image processing to reduce the amount of noise picked up. This step can be skipped, but you might run into issues further down the pipeline

- First, open the image and obtain a mask for the object by using OpenCV's inRange function in the HSV colorspace. Why HSV? HSV is able to define the color 'green' in a more intuitive way compared to the RGB color space. The following lines of code does that.
  HSV Range is from [60-S, 100, 50] to [60+S, 255, 255]
  - Where S is a fixed parameter called sensitivity, set at 15, can be changed so that it can pick more colors closely resembling green. Adjust this according to the camera you are using

- You should then obtain a mask that more or less looks like this (assuming the item is in the image)

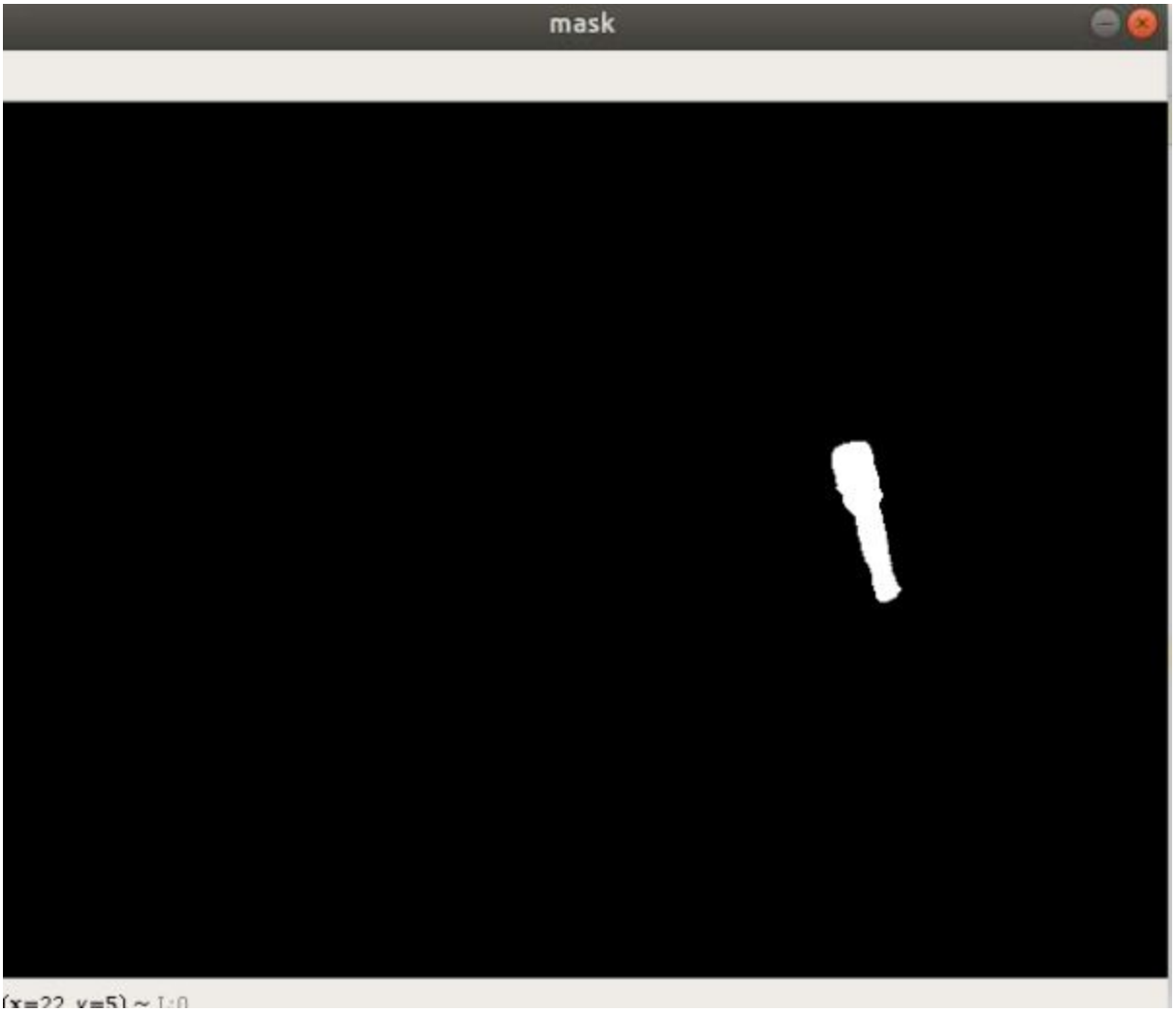

Figure 1: Object mask obtained by cv2.inRange Function

- Then, perform contour analysis on the mask image obtained from the above step using OpenCV's FindContours method using RETR_SIMPLE and using CHAIN_APPROX_SIMPLE. As the function says, it doesn't use anything complicated to measure and detect contours and thus should run quickly.

- Loop through this array of contours and then choose the contour with a significantly large area. Since the camera sampling resolution was at 640x480, we can measure the contour's area using OpenCV's in-built function to measure the surface area of the contour. Any contour smaller than 50 sq. units (OpenCV pixel units? Feel free to change this based on your camera noise) is ignored, and there should be only one

big contour. This contour is selected. IF there are more than two ‘big’ contours that are detected - then oh boy - this is where it gets difficult. We assume we haven’t found our object as according to our problem statement - we are detecting just ONE object. We will discuss this further down.

- Once this contour is selected, we obtain it’s x, y coordinates and the object’s width and height. Using this, we can draw a rectangle on our detected object in the image. (You need two xy coordinates for a rectangle in order to draw it - one for the top-left part and the other for the bottom right part, the coordinates for the latter can be obtained by adding the width and the height of the rectangle, i.e (x+w, y+h)

Figure: 2 : An image obtained by performing a Bitwise AND operation on the sample image and the mask obtained. You don’t need this really.

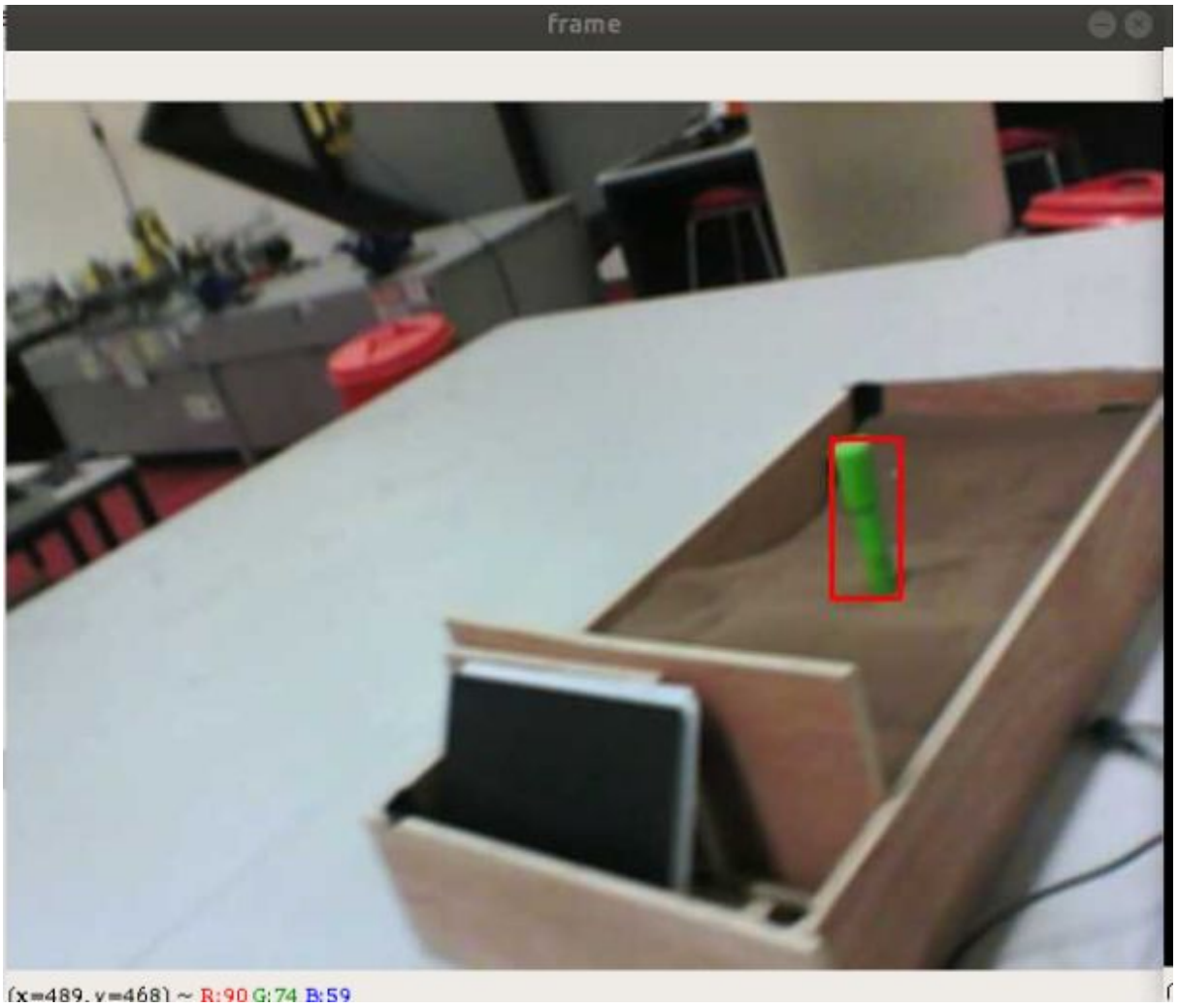

Figure 3: The detected object with the bounding box. This is more for visualisation and to 'show off' the fact that our idea "works".

Voila and there you have it! It's a simple and rudimentary strategy with great advantages and some disadvantages

Advantages of Color masking:

- Very fast, processes a 10-minute video on the LattePanda in 1 minute.
- Simple and elegant - easy to understand as well.
- Classifies the green object no matter what the angle or the position or the skew of the object is.
- Classifies the green object in sand too!

I strongly believe that the biggest advantage of this technique is that if used properly - is very efficient and can successfully provide a YES or NO answer as to whether the object is detected.

Disadvantages

- Sure, you can claim that you've made a very good object detector - but the truth is you've made something that can only detect objects that are green. **It is not a true "object" detector. Any presence of other green object will throw it off (aliens for example? Do they have aliens in Mars?)**

- Honestly speaking it can only provide a YES or NO answer as to whether there is a possibility that the object is in the frame.

However, not all is lost. We will use the above algorithm to pipeline our object detection strategy.
You see, what we have made is a method to detect and classify a green object. Assuming you control the environment in which it's taken - it does a great job of classifying the green "cache". It provides us with the bounding box. Think about it - you have a software that classifies the object (cache) for you and also hands you the bounding box on a silver platter.

So what's so great about this? We will find out in the next part

1.4 The Object-classifying Neural network

Ofcourse, we could use a neural network to classify the object. Surely it will do a better job than the method above, right? After all - a neural network is supposed to be "intelligent".

In order to train a neural network for classifying objects in images, one would have to obtain samples of what the object would look like and then train it on the neural net. Like a decent no. of samples. Our algorithm above just does that.

Rather than build a neural network from scratch, what I did was to use an existing neural network for object detection. The TensorFlow Object Detection API allows for reusing existing object classifiers built by Google using a technique called Transfer learning. As Google have trained a model that can detect a variety of objects, all we do is train our objects into the network and voila - we have an object detection neural network of our own!

TBC