Assignment 1: Extraction of Pixels of Ducks
Assignment Report

410421304 資工四 秘子尉

How to do the assignment?

In this assignment, I choose to use python as the programming language I am going to use for programing. It was hard from the very start, since I haven't use this programming language before this semester. Just like what everyone says, it is really easy to get started, but It can also contain multiple problems, such as problem due to not familiar with language, not familiar with support libraries, get confused with some grammar or syntax, etc.

At first I start my programming with simply build up several for loops, just like the way done before. Stack up one by one, finally finish building the probability function, then when I start my first approach, it was terrible. It just can't even run—the program I wrote was taking too much time doing calculation, and it did not finish after one night.

Then I was trying to make the program can produce at least some result first—so I resize the original class-provided test data into smaller one, so that the time my program working can be hugely decrease, although it still take several hours to done this, maybe either the code is too bad or I was working on some other thing at the same time to make computer to have huge burden to run the program.

As the picture below(picture 1), this was the first result of my program. At that time I used some special mechanism to collect the duck and nonduck data, just by simply using colors to draw on the sample picture, when my program scan through the picture and found specific colors, it will record the x and y transform of that pixel, then use that x and y we had found before to found the "origin" pixel on the origin sample image, then use this as the training data we are going to train our probability function, and finally scan through the sample input picture again to apply the function to each pixel—on each channel, I split them apart them combine the result at the end.

The result are bad, since there may be large bias when we drawing our example image, and also some pixel values may change when convert a picture to jpg file. When it goes to png file, the file size get increase a lot. And most importantly those pixel we mark on the example image were just bad, there were also some mistake in the function, so the program can only specific such as lake or grass, those ones which have huge pixel variation to the duck, but sand ground or rocks, they are simply just misclassify a lot.

Then I start to modify my first program. In the version 2 of the program, I have change some main part of the program, such as save some values that I had already use in case it will probably been used a lot in the loop, so that I can save some time to run the program. Combine some similar process to make the program lighter, and finally I can use the original image as the input train data, though it still need to take hours to run, but this time it got a much better result, since I have change the sampling method in this version. For the duck, I will the ducks I extract on the original image to a single color image. The program will skip those color and only focus on duck's pixel. So this time the result is much better compare to the first one, though still need improvement, since a huge part of rocks and grounds were misclassify as the duck.

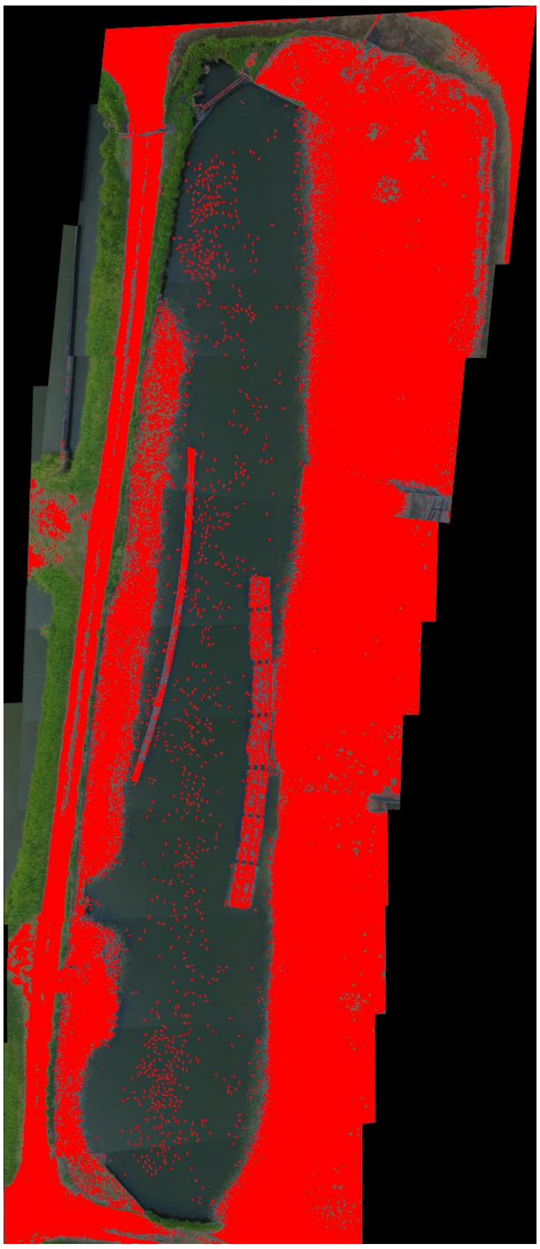
Then in the third version, I was been encouraged to use numpy function for array and matrix evaluation, so I use it for a huge part of the probability function, which help me improve the running speed a lot. And also in this version, the sample duck image were put on the above of pure white, said that to be [255,255,255]. And this method was a transform from the second version, which we had limited only very red(color we paint) were the duck pixel, means they got huge weight during training. In third version we skip the red-to-duck process, we assume that the greatest difference is that the duck can be pure white, but the rocks can't. The result were pretty good, in either speed or the outcome, it only took like 2000sec to run a single run.

After I got a good result, I were try some different approach. I was using my hand-build function for running, but not the build-in function. This time I try to use the multivariate method from scipy, unfortunately it seems that it take much more time then my third version approach, it takes like 2500 sec for the run.

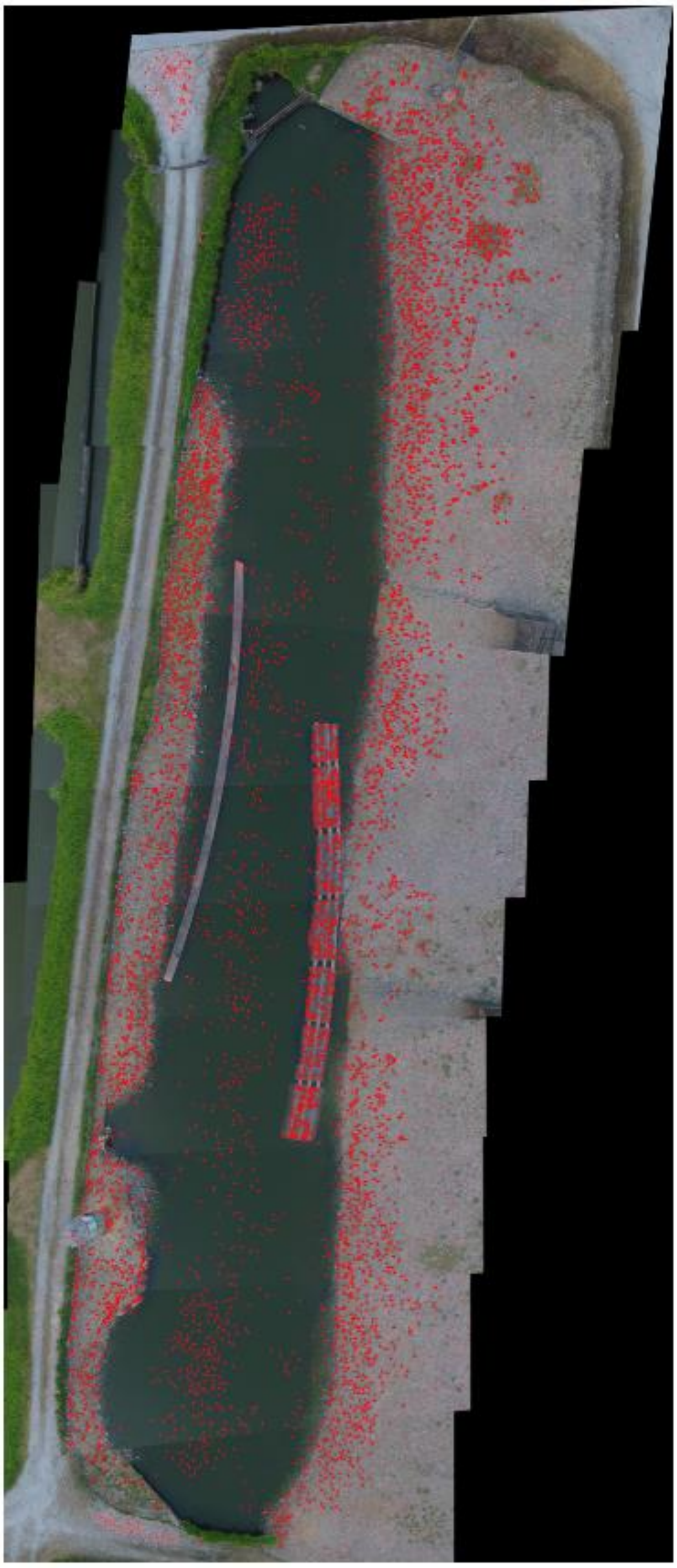
I had also try to use mask method, turning my result from the third outcome to black-white only image. Though the program is short, its till take me a lot of time learning these method, and the output looks cool.

The final thing I try to do is apply k-means method. I want to plot my result just like the output of the example given in class, give a single duck a dot on them, but unfortunately it was just half-finish. It took the last few days for my programing and it also need a lots of time for running the program, so the result were not that optimal for now, but it can filter those rocks.

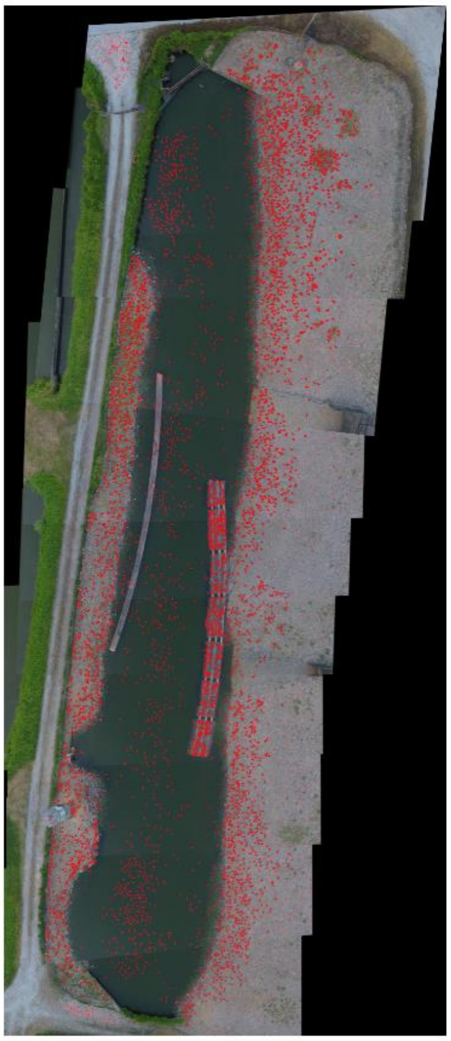
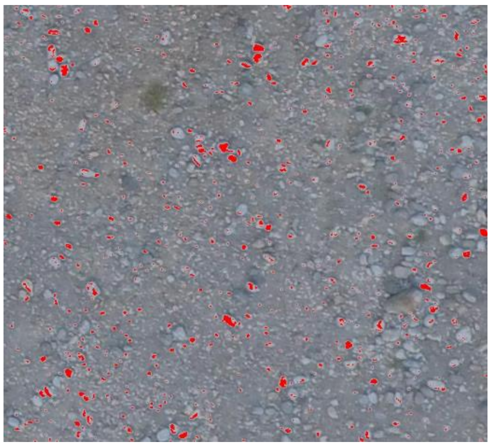
The results I get :



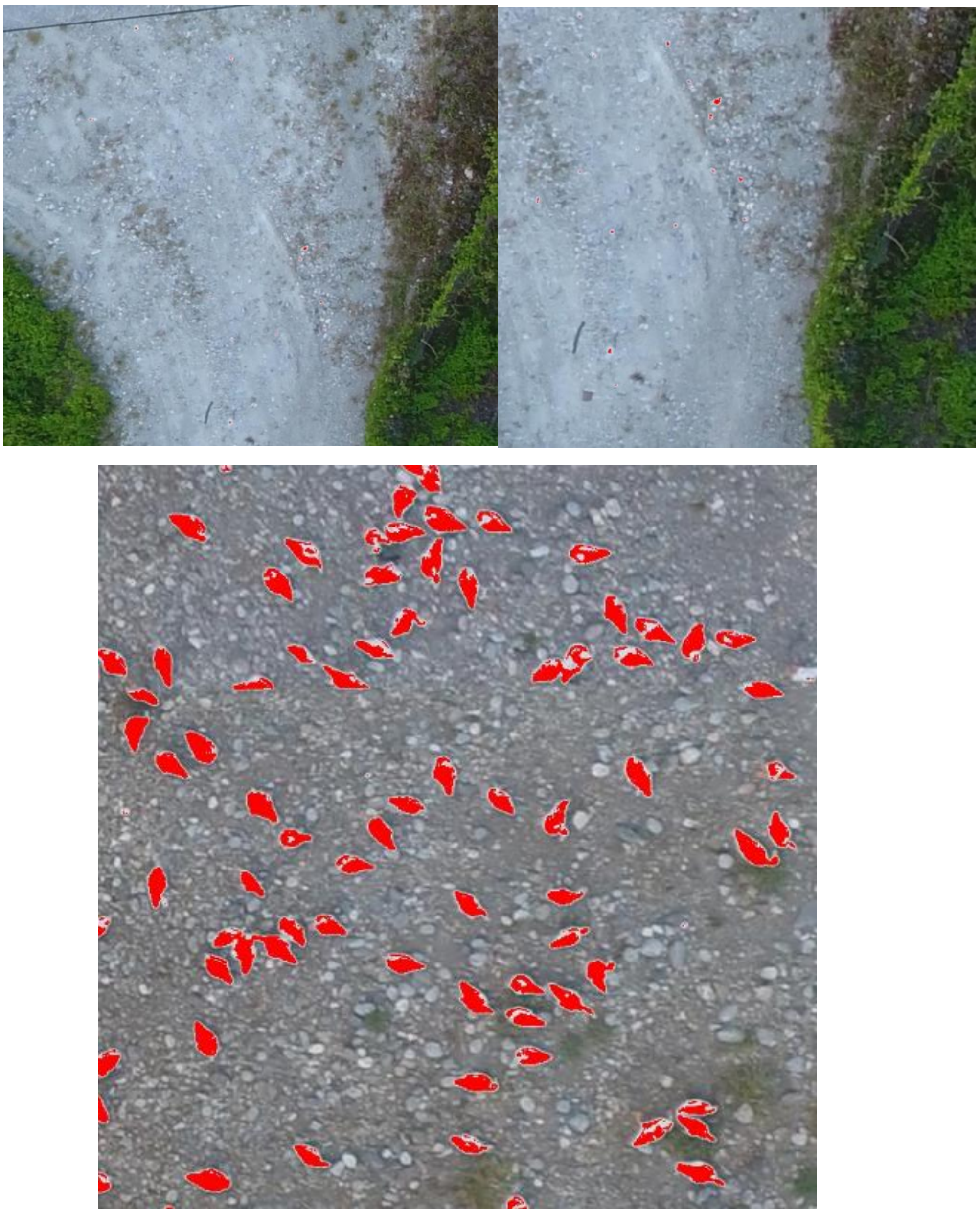
Picture one: the very first result. The program use express image as training data, and also there is some mistake in my program and sampling, so this result was really bad.

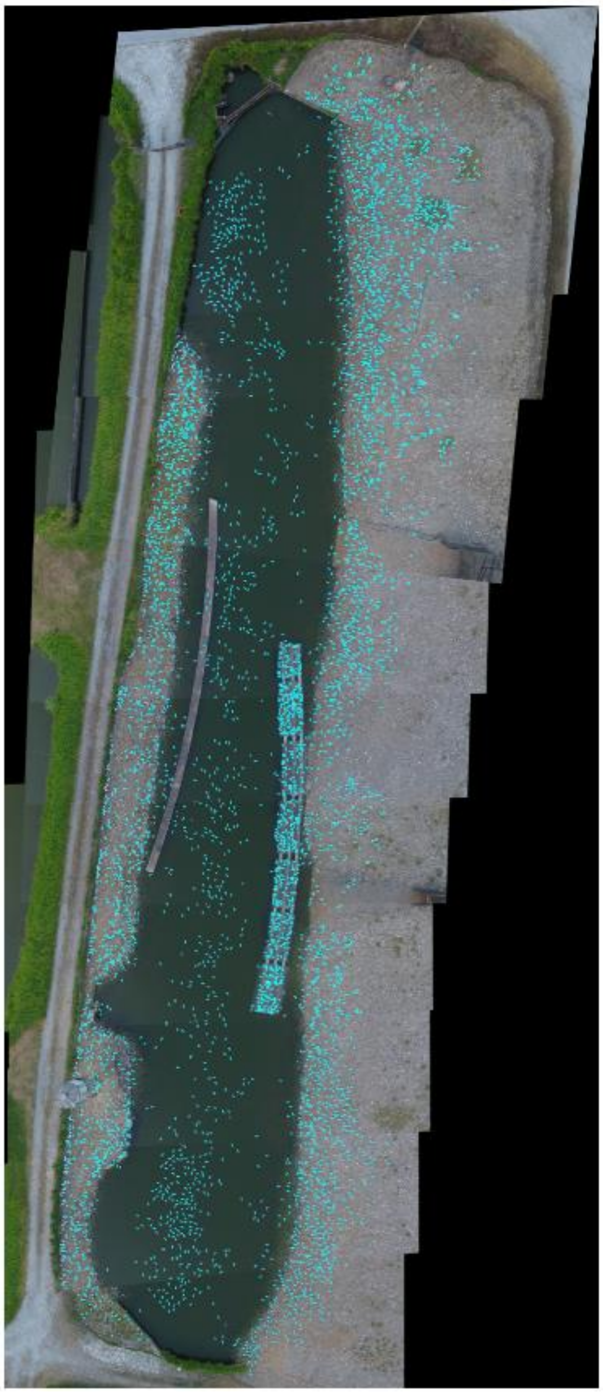


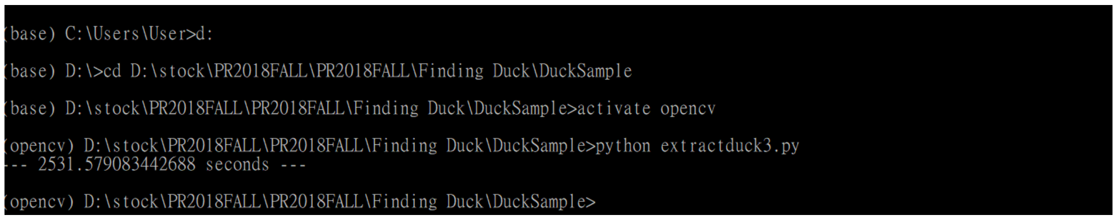
The picture of result of first version of program: In this version, there are a lots of outliers.

Picture 2: After some modify… The second version of program, after several changes, the grounds can be classify as non-duck for now but still lots of small rocks


Picture 3: Some trade-offs: In the later version of the program I have set some value filter to make only strongly relative points to be classify as ducks, however this may lead to some loss on ducks' body.

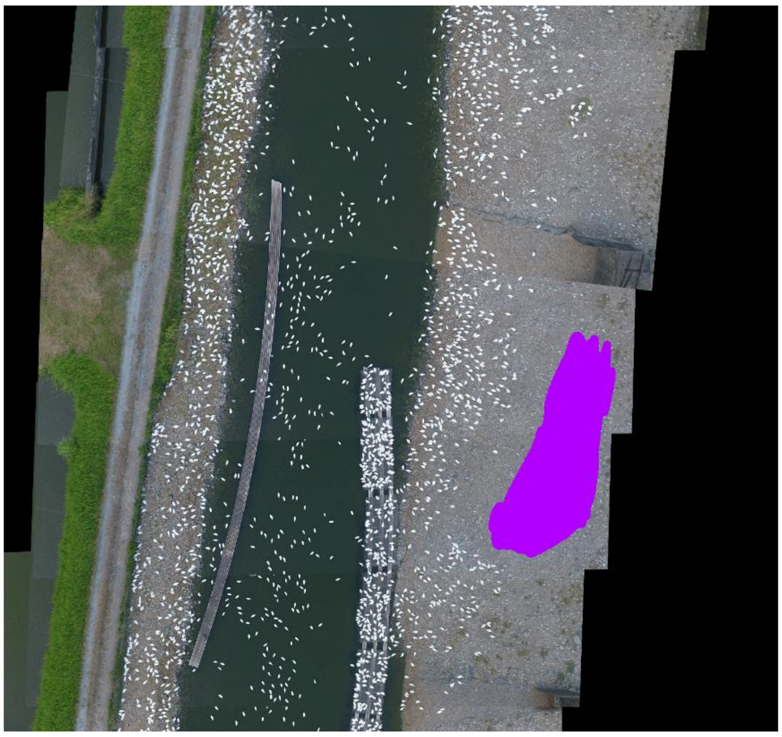


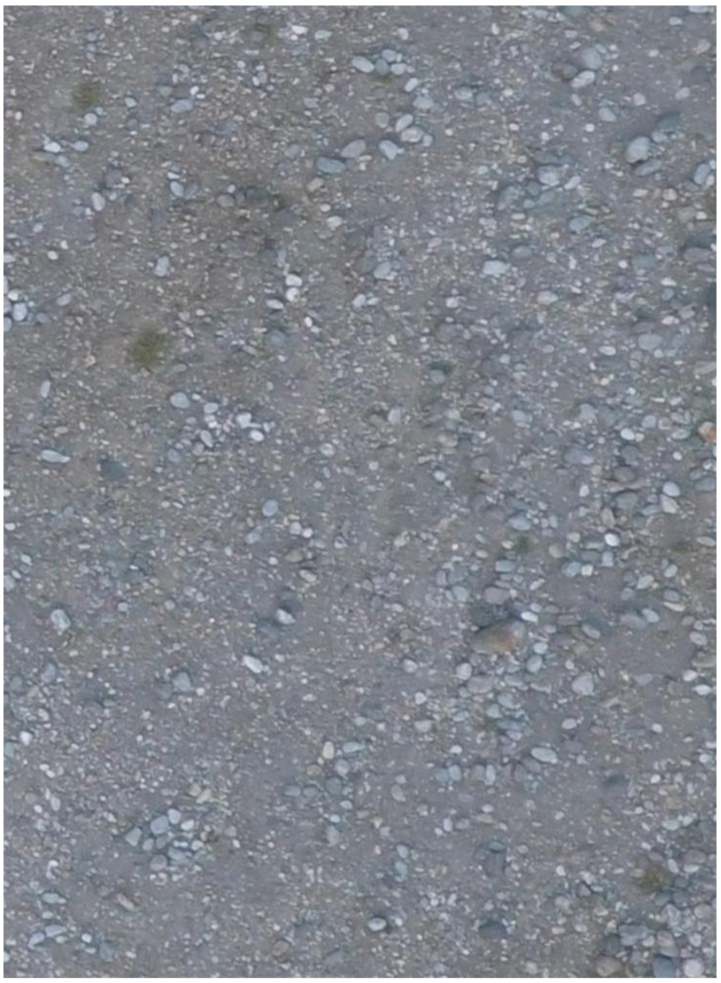


```
(base) C:\Users\User>d:

(base) D:\>cd D:\stock\PR2018FALL\PR2018FALL\Finding Duck\DuckSample

(base) D:\stock\PR2018FALL\PR2018FALL\Finding Duck\DuckSample>activate opencv

(opencv) D:\stock\PR2018FALL\PR2018FALL\Finding Duck\DuckSample>python extractduck3.py
--- 2531.579083442688 seconds ---

(opencv) D:\stock\PR2018FALL\PR2018FALL\Finding Duck\DuckSample>
```

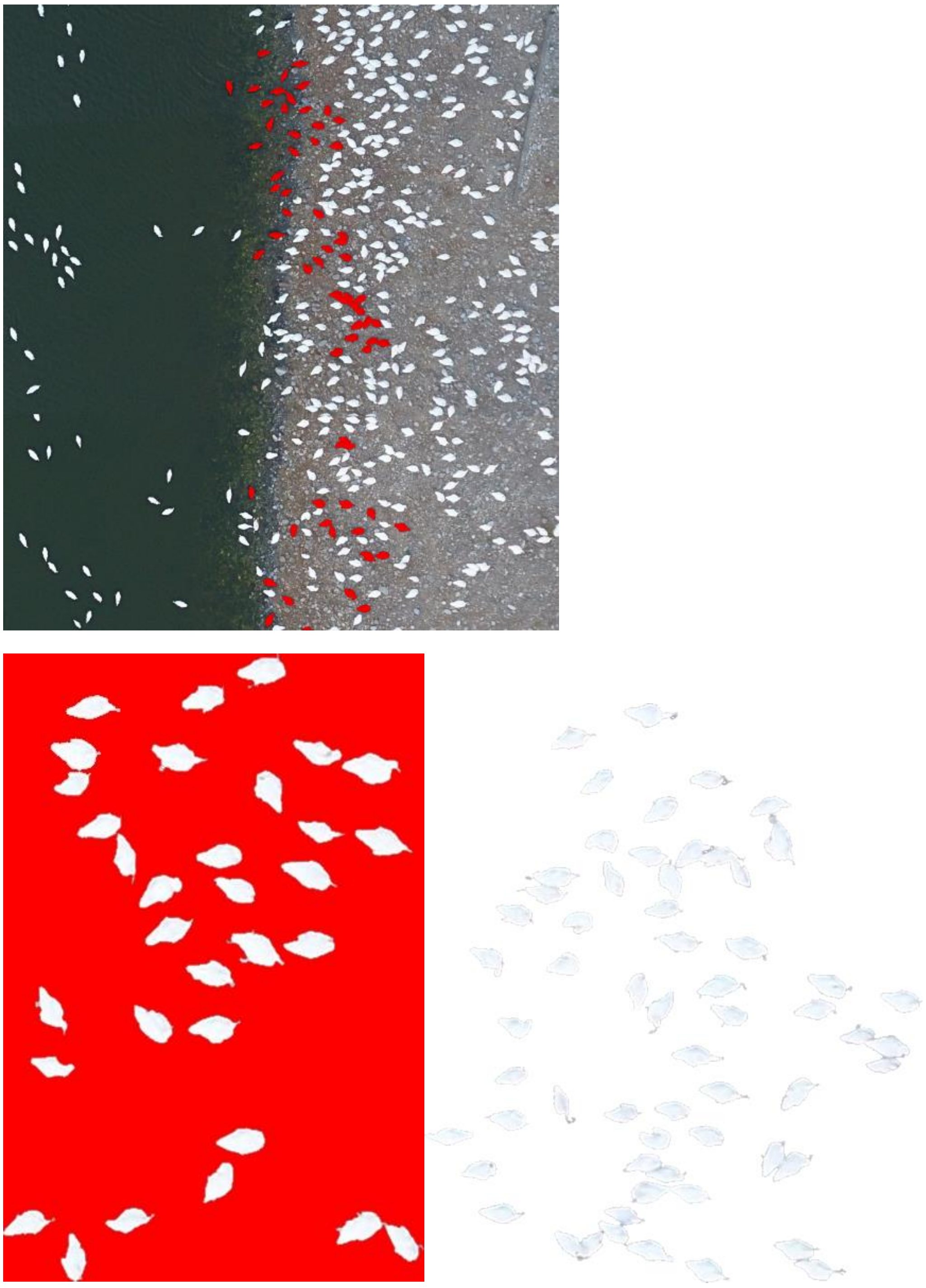
Picture 4: 3rd and the Best version, since we used some method on sampling, the most part of the rock can be filter by simply compare the probability of the function output.

Picture 5: Some other interesting work: mask

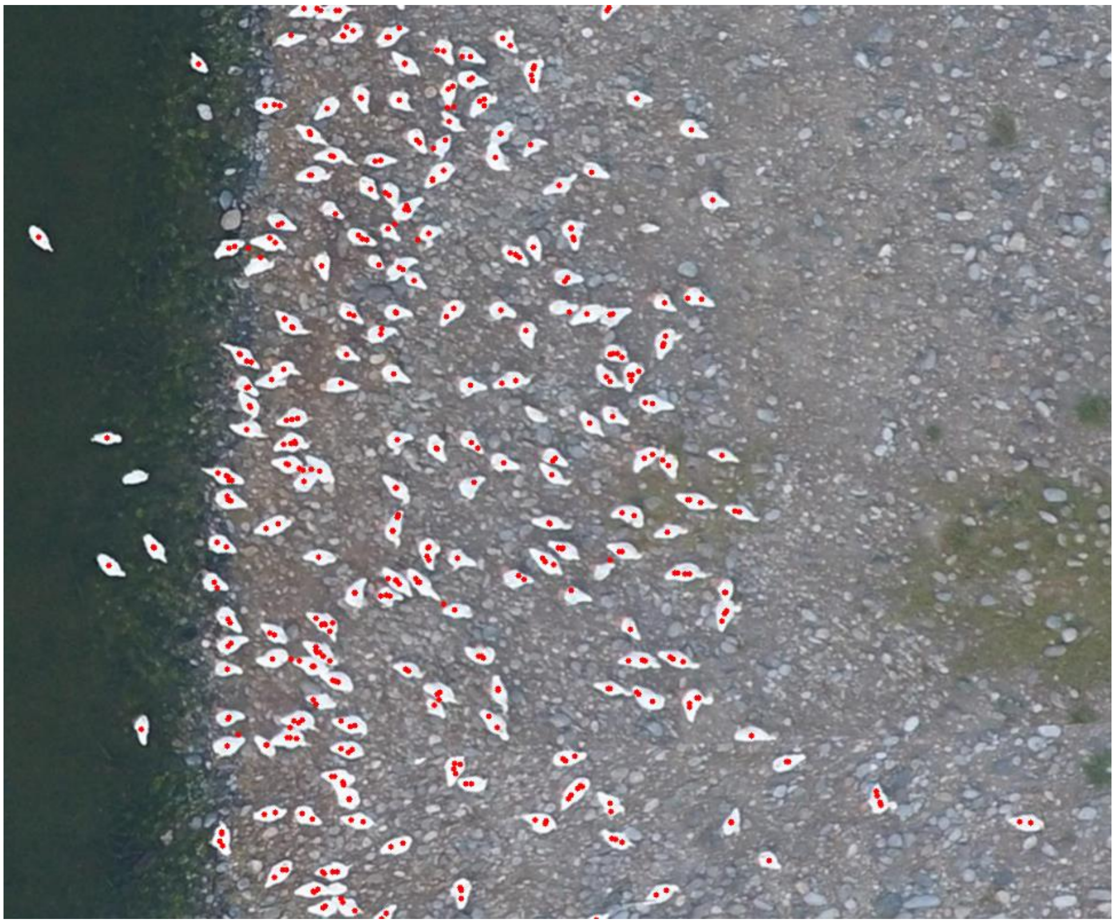


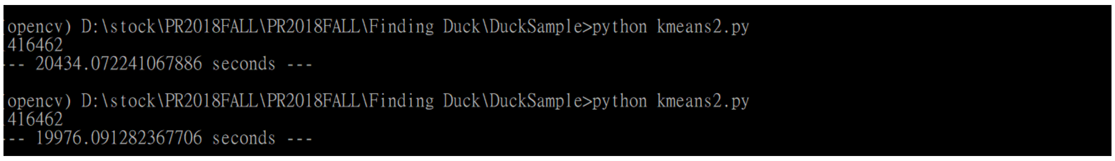


Picture 6: Different sampling method, above one were the original approach.



Picture 7: Different duck sampling method.





```
opencv) D:\stock\PR2018FALL\PR2018FALL\Finding Duck\DuckSample>python kmeans2.py
416462
-- 20434.072241067886 seconds ---

opencv) D:\stock\PR2018FALL\PR2018FALL\Finding Duck\DuckSample>python kmeans2.py
416462
-- 19976.091282367706 seconds ---
```

Picture 8: The result of k-means with dot count filter. The dot for a single duck is great, but seems the filter of multi-duck were wrong.

Discussions on the results:

As the statement we had given in the first two pages, and the picture we put above, we can found that the result were getting better and better.

In the first few approach, we use some bad sampling method, just simply draw a huge area manually, this is dangerous since that we can make some mistake during drawing, and also it takes a lots of time for program to re-processing those un-use pixel. So for the non-duck sample, we choose the hardest part we had met: the ground with rocks. We choose the area from the first few misclassify part as the new input, and for ducks we make every pixel been useful: either been a filter, or just scratch some important part out of the duck and use that color as the background, just as set huge weight for that, and by these alternative bootstrap method, we can see on the third version result, it was huge improvement compare to the first one, In this part the sampling take important rules.

Beside the performance, another important thing is the running time. From the beginning, my program couldn't even finish after one night, which make me understand that brute force won't get good result for a huge amount of data, so I try to improve my program. First, decrease the amount of calculation. Set some variables or arrays to store the result we had made before, to avoid the unnecessary re-calculation. Second, try to avoid loops. Loops take a lots of time in programing, especially when you have huge image to process. I had try the recursive method in some version, but since the recursion were too big(like 8 thousand thousand), it just simply stack overflow, so I go back to read some tutorial, use numpy as the method to do matix and/or array processing instead of access every pixel directly. This save a lots of time.

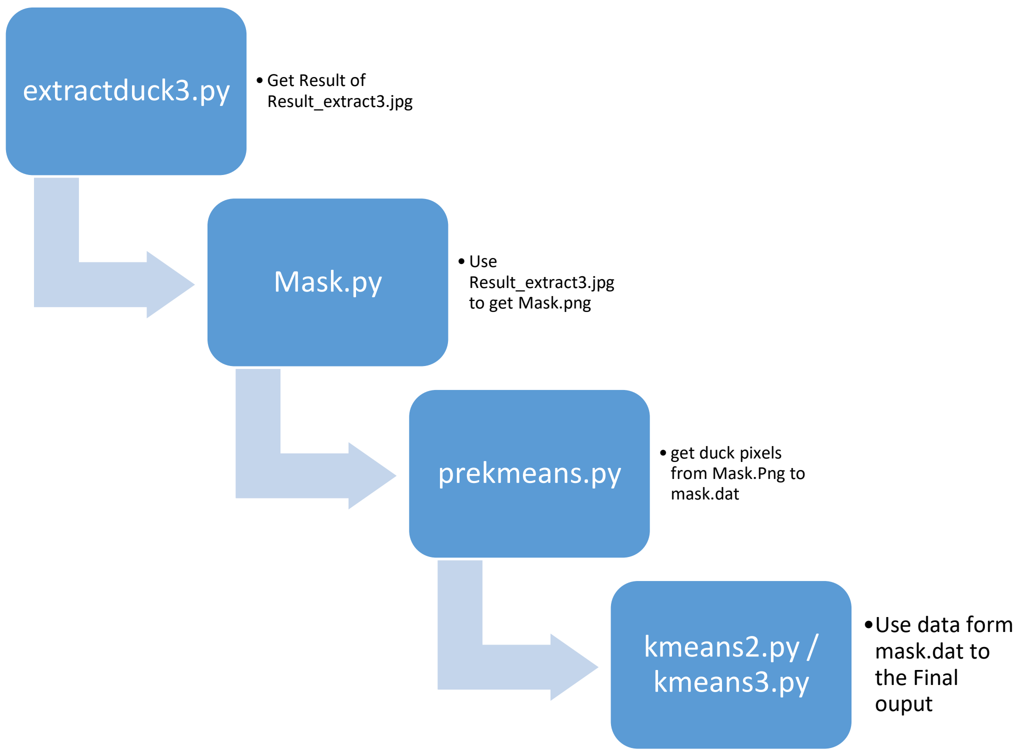
For example, in mask.py, I use numpy method to let the arrays compare themselves to see if it have a specific feature(looks blue), instead of just go through every pixel, which can save huge amount of time to process the loop.

The way I improve my program and result is by try and error. Since you will never know whether this kind of method would provide you a better result, I would try everything I can for the best result. For example the sampling, although using filter must seems to be the best way for get a good sample set, but you just needs to try other first so that you can see the improvement. And the bootstrap, by changing the non-duck training set to those we had misclassify to see which feature could separate duck and non-duck pixels better, I could finally got a well-look result. This could easily been seem visually: first we had misclassify lake, so I choose lake for dataset(not list above, only in initial test), then when it comes to first approach, it seems that those darker pixel does not affect the prediction so much, the important part is the ruck, since it has just a little variation to the duck(on pixel values), it could even been hard for human-beings to classify if it were duck or rock. But since we know that rock is small, duck is big, so I had made a k-mean program, in this program it would do the post-process to the image, combine with filter those small group of outlier(the amount of pixels in a small area), and also make the output looks better, for each duck we give it a dot, in optimal.

But unfortunately I cannot finish this part on time, but I am glad that at least I try this method and finish the mask part. The best result I got shows that this k-means method could plot single duck perfectly, almost plot the dot in the center, but the program still need to improve the factor to filter multi-duck from single ones, and also need to plot different colors for multi-duck for better way to visualize the result.

The most things that affect my output result is definitely factor and the program structure. From this scenario we could know that the way I improve my program is first thought a new method I am going to achieve, then check the performance of the program and then improve it. By these steps I can make my result better and better.

Current Program Structure:


extractduck3.py
• Get Result of Result_extract3.jpg
Mask.py
• Use Result_extract3.jpg to get Mask.png
prekmeans.py
• get duck pixels from Mask.Png to mask.dat
kmeans2.py / kmeans3.py
•Use data form mask.dat to the Final ouput

Summary:

In this assignment I have learn a lot about python, especially the part to press list, array, or ndarray. My programming skill was not really strong since before, so mostly I would try the method I thought by myself, then compare to others, ask some question and try to improve and / or found a new method then try to finish it.

I also learn some new image process method, about opencv usage, and like k-mean, though I had not get optimal solution yet, during the programing I think I had get a lot more familiar with this method.

During this practice, I think myself still lack a large amount of programing experience, although I can come up with good ideas or approach, but since the dirty codes some method could have been unable to realistic at this moment. I need to try to improve the quality of the code in case when I face a much larger work in the future, it could just be a nightmare since nothing could run under those unnecessary process.

My thought about this assignment is that this is really interesting, although there may be some frustration such as bug code not be solved, hard to optimize the code, and it also take me a lot of time to do this assignment to make it better and better which make me really tired, I certainly learn a lot of thing during programing, and also searching those issue and solution, and discussion with others to make both of us to have greater result. Having these kinds of experience, it could be a great help in my future of programing.

Appendix:

Link of Programs (Current Version):

Note that old the old versions were store in the old version file storage.
Version 3 of Program:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/extractduck3.py

Mask:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/mask.py

Pre-Kmeans:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/prekmeans.py

Kmeans2:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/kmeans2.py

Kmeans3:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/kmeans3.py

Results:

From Version3:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/Result_extract3.jpg

From Mask:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/Mask.png

From Kmeans2:
https://github.com/ndhu410421304/PR2018FALL/blob/master/Finding%20Duck/Duc
kSample/KMean2.png