Submission for Summer 2019 Assignment Evaluation

SKYLARK™ DRONES

Automatic GCP Detection in Drone Imagery using Computer Vision

Tanuj Vishnoi,
Former Intern, Nvidia Labs, New Delhi
Thapar Institute of Engineering and Technology

1. Problem statement

Manual detection of Ground Control Points is a cumbersome task and thus, a Computer Vision and Deep Learning model can help simplify this task by identifying the points and then plotting their coordinates, since the objects of detection GCP are very small and human findings can take a lot of time.

Problem with using object detection: The GCP points are very small and thus, normal object detection algorithms could not be applied. There is scope of applying **RetinaNet model**, but that has led to bad accuracy, and great computation since it uses detection.
So, basically it's a classification problem for detecting if the given object is the required "L" shaped GCP or not.

So the required characteristic of problem and output is:

Fig 1:Detection of GCP point, which was otherwise not clearly identifiable from naked eyes

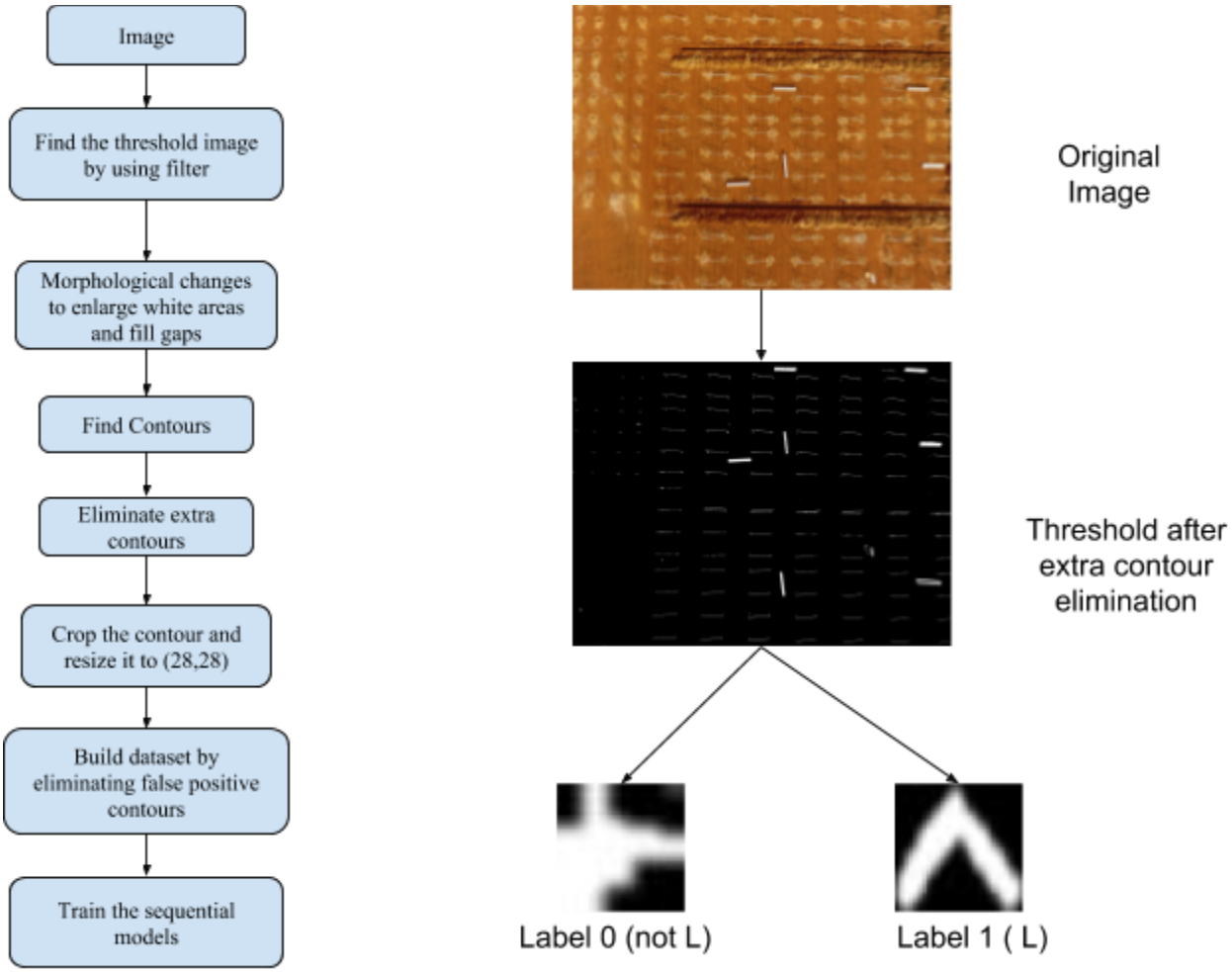
2. Methodology Adopted

Computer Vision powerful tool opencv allows image modifications to get desired data. Here are the function used.
Refer these files image_process.py and dataset_prep.py

1. The kernel sizes are variable and are suited on test and trial bases.
2. cv2.threshold : Since the GCP points are white, so a thresholding between (220,255) give a better black-white presentation of image for contour plotting.
3. cv2.morphologyEx: The closing and opening are used for enlarging white areas by adding pixel (Dilation) and then noise removal by adding pixel (erosion) to the above obtained binary image.

4. cv2.findContours: Used for finding contours in threshold image for shape identification of GCP, “L”.
5. Then extra contours are eliminated by checking concavity and area of each contours. The area threshold values are hard-coded and can be changed.
6. The required contours are found by calculating the closest distance from the labelled coordinates and it is cropped and stored as label 1. Rest rejected contours are also cropped and stored as label 0.
7. The contours are cropped not as horizontal rectangles but as the rotated best fit rectangles and are then cropped accordingly.
8. The dataset is then manually filtered for wrong values.

Below is the flow chart of the methodology followed during training



3. Neural Network Architecture (For training):

Refer this notebook: training.ipynb (Github)

For training I have used a sequential model, since now the dataset is just like MNNIST dataset and the network does not need much larger convolutional layers, however, models like VGG16 can be used for the same. The implementation is done in keras. The learning model is a Sequential model with 5 convolutional layers build in keras. The loss chosen in categorical cross-entropy. However other losses like rmsprop may be used. The optimizer chosen is Adam.

To get a single max probability output, softmax is used. The filters kernel size in the convolutional layer are on trial and test basis. The training data has 286 positive samples and 213 negative samples contained in repository 'data/train'. The test samples has 8 samples each of negative and positive L, contained in repository 'data/test'. 70 epochs showed best accuracy, else the model was getting overfitting. The model summary is as follows:

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d_6 (MaxPooling2	(None, 28, 6, 7)	0
conv2d_7 (Conv2D)	(None, 28, 6, 50)	8800
max_pooling2d_7 (MaxPooling2	(None, 28, 2, 10)	0
conv2d_8 (Conv2D)	(None, 28, 2, 80)	12880
max_pooling2d_8 (MaxPooling2	(None, 28, 1, 16)	0
conv2d_9 (Conv2D)	(None, 28, 1, 100)	25700
max_pooling2d_9 (MaxPooling2	(None, 28, 1, 20)	0
conv2d_10 (Conv2D)	(None, 28, 1, 100)	18100
max_pooling2d_10 (MaxPooling	(None, 28, 1, 20)	0
dropout_3 (Dropout)	(None, 28, 1, 20)	0
flatten_2 (Flatten)	(None, 560)	0
dense_3 (Dense)	(None, 512)	287232
dropout_4 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 2)	1026

Total params: 354,570
Trainable params: 354,570
Non-trainable params: 0

Required CSV for training are data1.csv (Github) and data2.csv (Github) for ML-Dataset#1 and ML-Dataset#2 respectively. The training csv look like this (beautified version as on github)

ML-Dataset#2/M1_F1.3_0403.JPG	704	656
ML-Dataset#2/M1_F1.3_0404.JPG	685	1066
ML-Dataset#2/M1_F1.3_0405.JPG	664	1486
ML-Dataset#2/M1_F1.3_0406.JPG	648	1908
ML-Dataset#2/M1_F1.3_0407.JPG	634	2339
ML-Dataset#2/M1_F1.3_0408.JPG	625	2787

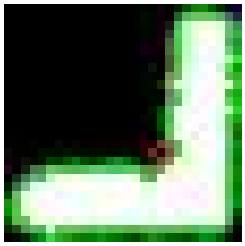
The keras model is saved as hdf5 file.

4. GCP Extraction: the Output

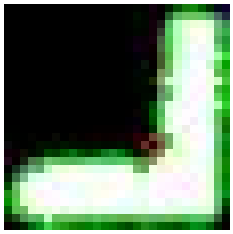
Refer this notebook: gcp_extraction.ipynb (Github)

The image is pre-processed as before for all the contours and then eliminating extra contours from the output contours. The accepted (required contours) are then passed to the model for prediction. If the output is 1, i.e. L is detected follows happens:

1. The coordinates of the bounding box in the original image are captured using opencv drawContours function and stored in crop variable. The filename, threshold image, output required contour lines and crop are then returned as a list from the plot function.
2. On analysis it shows that the required point in **L GCP is always near the Center of Mass of the L.**



3. Then corner points are detected in the threshold image using **Harris Corner detector.**
4. The Harris corner points are dilated and then passed to a cv2.cornerSubPix() function to get corners with Sub Pixel Accuracy (Source: link)



5. The closest point of the harris corner with the center of mass is the required point, whose coordinates are recorded in output.csv (Github), which are in format as in SampleOutput.csv file.

FileName	GCPLocation
DSC08876.JPG	[[4164.902 171.14754]]
DJI_0422.JPG	[[2609.2104 2253.8484][507.13623 1613.8837]]
DJI_0422.JPG	[[507.13623 1613.8837]]
DJI_0083.JPG	[[2458.4124 2062.5876][2335.0178 573.03754]]
DJI_0083.JPG	[[2335.0178 573.03754]]
DSC01453.JPG	[[3128.804 1170.9548]]
M1_F1.3_0405.JPG	[[982.9792 2688.448][665.07776 1486.0809]]
M1_F1.3_0405.JPG	[[665.07776 1486.0809]]
DJI_0617.JPG	[[3780.3523 33.220127]]
DJI_0036.JPG	[[1742.4098 2597.1355]]
DJI_0616.JPG	[[254.71074 1455.9338]]
DJI_0086.JPG	[[2528.279 2445.4182][1474.6449 1496.343][2360.0862 777.64453][2284.9143 124.52857]]
DJI_0086.JPG	[[1474.6449 1496.343][2360.0862 777.64453][2284.9143 124.52857]]
DJI_0086.JPG	[[2360.0862 777.64453][2284.9143 124.52857]]
DJI_0086.JPG	[[2284.9143 124.52857]]

5. Further Improvements

1. Provided with limited time, I didnt get chance to explore further models. Models like VGG16, Inception and more can be tested. Further model with more convolutional layers can be tested with different filter sizes.
2. The dataset can be improvised with more training images.
3. There have been some false predictions on trucks, and so objects like these can be detected and eliminated from threshold images.
4. Calculations on angles can be worked on to get more exact co-ordinates.