Abstract

Detecting a horizon in an image is an important part of many image related applications such as detecting ships on the horizon, flight control, and port security. Most of the existing solutions for the problem only use image processing methods to identify a horizon line in an image. This results in good accuracy for many cases and is fast in computation. However, for some images with difficult environmental conditions like a foggy or cloudy sky these image processing methods are inherently inaccurate in identifying the correct horizon.

This investigates how to detect the horizon line in a set of images using a machine learning approach. Accurate horizon detection is essential for a number of applications such as port security, flight and navigation control, flow management, port safety, and protection of sanctuaries One of the previous approaches used only image processing methods, which, however, performed very well in most cases. Another approach was combining machine learning methods with morphology based operations, the Hough Transform and Expectation Maximization function to separate pixel distributions. However, their performance in identifying a horizon suffers when images are complicated with clutter such as a very cloudy or foggy environment, an uneven horizon line, and varying lighting conditions. Partly this is because features used in the approaches only included only intensities of the pixels in an image, which is not enough for identifying horizon in “hard” images. Method used in utilizes texture information in images.

Contents

Chapter 1

INTRODUTION

Detecting a horizon in an image is an important part of many image related applications such as detecting ships on the horizon, flight control, and port security. Most of the existing solutions for the problem only use image processing methods to identify a horizon line in an image. This results in good accuracy for many cases and is fast in computation. However, for some images with difficult environmental conditions like a foggy or cloudy sky these image processing methods are inherently inaccurate in identifying the correct horizon. This paper investigates how to detect the horizon line in a set of images using a machine learning approaches.

Accurate horizon detection is essential for a number of applications such as port security, flight and navigation control, flow management, port safety, and protection of sanctuaries.One of the previous approaches used only image processing methods, which, however, performed very well in most cases. Another approach was combining machine learning methods with morphology based operations, the Hough Transform and Expectation Maximization function to separate pixel distributions. However, their performance in identifying a horizon suffers when images are complicated with clutter such as a very cloudy or foggy environment, an uneven horizon line, and varying lighting conditions.

Detecting the horizon can be seen as finding a line which separates the image into two regions: sky and non-sky. Non-sky pixels usually belong to the ground or water surfaces, and, to be consistent with many papers which refer this problem, the non-sky pixels will be called further as *Ground* pixels. Thus, one of the objectives of the work in this paper was to attempt to successfully classify the pixels of the image into two major classes: either the *Sky* class or the *Ground* class, by using the selected classifiers.

Once we could classify each pixel into two classes, we could create a binary image, where a pixel with value of “0” belong to the *Ground* class and a pixel with a value “1” belong to the *Sky* class. Further in the paper such kind of an image will be referred as a black and white image where all the white pixels belong to the sky and all the black pixels belong to the ground. For this paper, an assumption was made that the line that would separate the sky and the non-sky pixels would be a straight line, and, thus, the other objective of the paper becomes finding a line, between black and white regions in the black-and-white picture, which would correspond to the actual horizon line in the original image.

Chapter 2

PROOF OF CONCEPT

The project called “Horizon Detection of Maritime Images Using Machine Learning Techniques ” is based on two papers.Fast horizon detection in maritime images using region-of-interest Asim M. El Tahir Ali, Hussam M. Dahwa Abdulla, Vaclav Snasel In this papper, we propose a fast method for detecting the horizon line in maritime scenarios by combining a multi-scale approach and region-of-interest detection. Recently, several methods that adopt a multi-scale approach have been proposed, because edge detection at a single is insufficient to detect all edges of various sizes. However, these methods suffer from high processing times, requiring tens of seconds to complete horizon detection.

Horizon detection based on sky-color and edge feature Bahman Zafarifar,Hans Weda. In this papper, they propose an algorithm for detecting the horizon line in digital images, which employs an edge-based and a new color-based horizon detection technique. The color-based detector calculates an estimate of the horizon line by analyzing the color transition in the clear sky areas of the image. The edge-based detector computes the horizon line by finding the most prominent line or edge in the image, based on Canny edge detection and Hough transformation.

Also referred website "Obstacle Detection for Small Autonomous Aircraft Using Sky Segmentation".

2.1 OBJECTIVES

In this project input is a maritime image data set that stored in the directory. we need to find out the horizon of that particular images using machine learning approaches. Canny Edge Detection is a popular edge detection algorithm.It is a multi-stage algorithm and we will go through each stages. The Hough Line Transform is a transform used to detect straight lines. To apply the Transform, first an edge detection pre-processing is desirable.The objectives of the project “Horizon Detection Using Machine Learning Techniques”are as follows. We are needed to implement the following aspects:

- 1. Given an image data set that contains set of Maritime images.
- 2. Images are preprocessed.
- 3. converting RGB.
- 4. finding best line using HoughLine transform and canny edge detection .
- 5. A single line which separates the sky and the non sky.

Chapter 3

IMPLEMENTATION

This project Aims to Detecting the horizon can be seen as finding a line which separates the image into two regions: sky and non-sky. Non-sky pixels usually belong to the ground or water surfaces, and, to be consistent with many papers which refer this problem, the non-sky pixels will be called further as *Ground* pixels. Thus, one of the objectives of the work in this project is to attempt to successfully classify the pixels of the image into two major classes: either the *Sky* class or the *Ground* class, by using the selected classifiers. Once we could classify each pixel into two classes, we could create a binary image, where a pixel with value of “0” belong to the *Ground* class and a pixel with a value “1” belong to the *Sky* class.Accurate horizon detection is essential for a number of applications such as port security, flight and navigation control, flow management, port safety, and protection of sanctuaries.

In this project I have investigated usability of different machine learning algorithms for horizon detection problem. Specific approach described in this project allows the identification of a horizon line in single images for the most adverse conditions of the horizon by using machine learning to provide a general model. It uses texture information along with pixel colors for its feature set. The approach provides accuracy of identifying horizon line within 90-99 percentage.

One of the previous approaches used only image processing methods, which, however, performed very well in most cases. Another approach was combining machine learning methods with morphology based operations, the Hough Transform and Expectation Maximization function to separate pixel distributions. However, their performance in identifying a horizon suffers when images are complicated with clutter such as a very cloudy or foggy environment, an uneven horizon line, and varying lighting conditions. Partly this is because features used in the approaches only included only intensities of the pixels in an image, which is not enough for identifying horizon in "hard" images.

Preprocessing of the given text files is the first module in this project. To preprocess the text simply means to bring the text into a form that is predictable and analyzable for the task. A task here is a combination of approach and domain .In natural language processing the preprocessing of the text data is done using various methods.Raw data contain numerical value, punctuation,special character etc as shown in Figure 1 and Figure 2 . These value can hamper the performance of model so before applying any text featurization first we need to convert raw data into meaningful data which is also called as text preprocessing . This can be done by following ways:

Step 1 : Data Preprocessing
- preprocess the image dataset
- convert images into grey scale.
- convert images into RGB format.

The image preprocess Success in classifying image pixels depends on the amount of training and variety of data on which classifiers get trained. To improve overall performance, number of attributes can be increased by using different sizes of regions around each pixel while calculating texture measures. It is also possible to introduce attributes based on the relative position of a pixel, so that we could avoid situations when we encounter *Sky* pixels among *Ground* pixels and vice versa. In the future feature selection methods can be applied to reduce number of features and improve the algorithm performance. In addition, utilization of additional pre- and post-processing can further enhance the performance.

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and we will go through each stages.
Step 2 : Canny Edge Detection
• Noise Reduction
Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.
•Finding Intensity Gradient of the Image
Smoothened image
•Non-maximum Suppression
After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.
•Hysteresis Thresholding
This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, min-Val and max-Val. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges. Otherwise, they are also discarded.

3.1 SYSTEM ARCHITECTURE

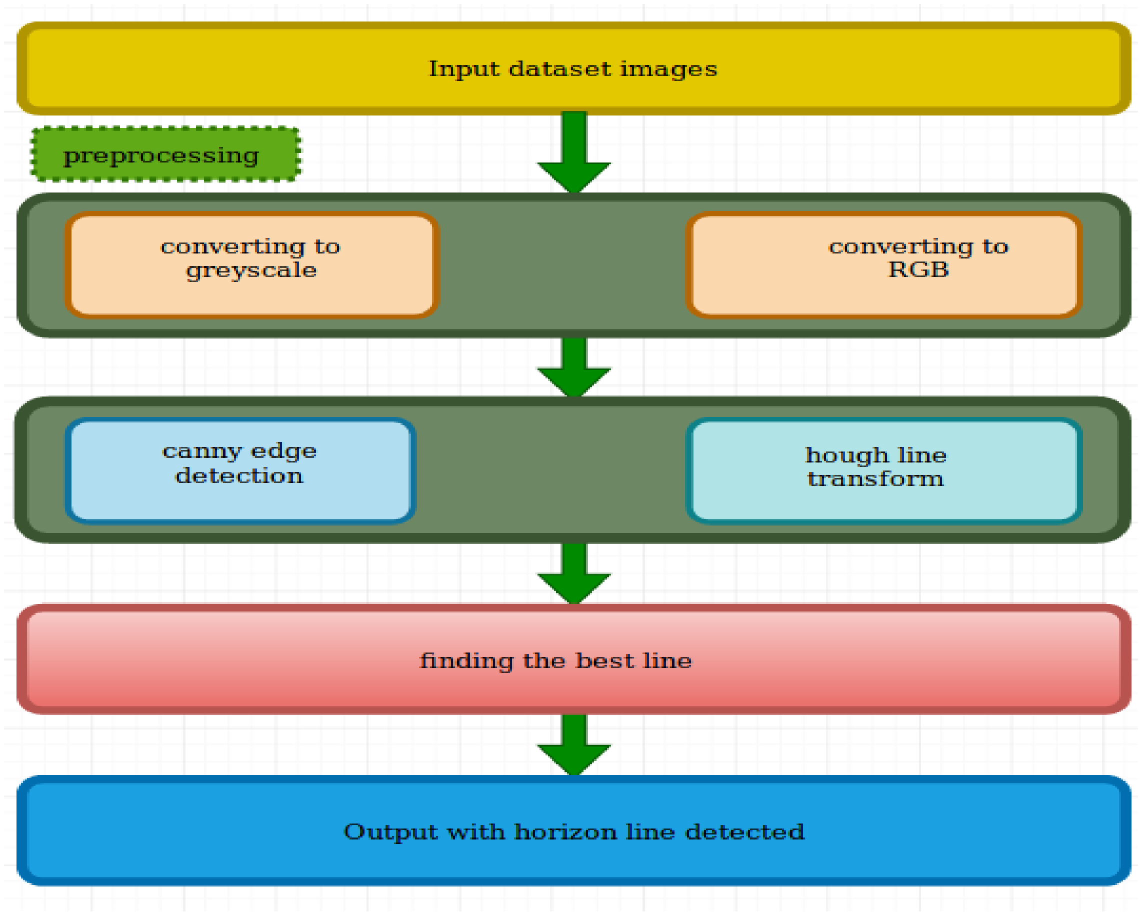


Figure 3.1: System Architecture

The figure 3.1 shows the overall working of the system.first off all we are inputing an image data set . And the preprocessing has to be done in the given dataset.converting the image into RGB and Grey Scale.From the preprocessed images we used the 2 process that is canny edge detection and the HoughLine transform. That will find out the edges and pixels and help to find out the horizon .hough line transform is helps to finding a line in the images. And finally a single line which seperates both sky an the non sky. That is the horizon which seperates the sky and the sea. the horizon can be seen as finding a line which separates the image into two regions: sky and non-sky. Non-sky pixels usually belong to the ground or water surfaces, and, to be consistent with many papers which refer this problem, the non-sky pixels will be called further as *Ground* pixels. Thus, one of the objectives of the work in this paper was to attempt to successfully classify the pixels of the image into two major classes: either the *Sky* class or the *Ground* class, by using the selected classifiers.

Chapter 4

ALGORITHMS USED

4.1 canny edge detection and HoughLine Transform

Edge detection and Hough transform-based algorithm (H-HC) works in four stages; first, the original image is filtered to remove slight distortions of high frequency, and erosion or low pass filter is used for that purpose; then, Canny filter is applied to extract strong edges in the image; next, straight lines are detected by means of Hough transform , and finally, the horizon line is identified as the longest line from those previously detected. Like the previous method, also H-HC is computationally complex; the cause is the Hough transform in this case.

The main idea of edge detection and least-squares calibration-based algorithm (H-LSC) is to determine maximal vertical local edges in each column of the image and then to find the optimal horizon line by the least-squares method. To remove small distortions of the image and then outliers, that is, maximal column edges which "deform" the optimal horizon line, morphological erosion and median filter are used in separate stages of the algorithm. In general, such an approach seems to be faster than the previous ones; however, applying the least-squares method, searching each column, pixel after pixel, for vertical edges and double use of morphological filters makes also H-LSC rather improper for on-the-fly processing in maritime conditions, especially, for high-resolution images and standard computer hardware used to process them.

.

Chapter 5

MODULES

5.1 Modules

5.1.1 Preprocessing

Image Preprocessing is the first module in this project.The image preprocess Success in classifying image pixels depends on the amount of training and variety of data on which classifiers get trained. To improve overall performance, number of attributes can be increased by using different sizes of regions around each pixel while calculating texture measures. It is also possible to introduce attributes based on the relative position of a pixel, so that we could avoid situations when we encounter *Sky* pixels among *Ground* pixels and vice versa. In the future feature selection methods can be applied to reduce number of features and improve the algorithm performance. In addition, utilization of additional pre- and post-processing can further enhance the performance.The steps involved in Image processing are:
Step 1 : Image Preprocessing
• converting the image dataset into greyscale and RGB
In python Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract

some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two dimensional signals while applying already set signal processing methods to them.

• Removing noicy datas.

• find out the image pixels

A pixel is the smallest unit of a digital image or graphic that can be displayed and represented on a digital display device. . Pixels are combined to form a complete image, video, text or any visible thing on a computer display.

• Detecting the Edges from the images

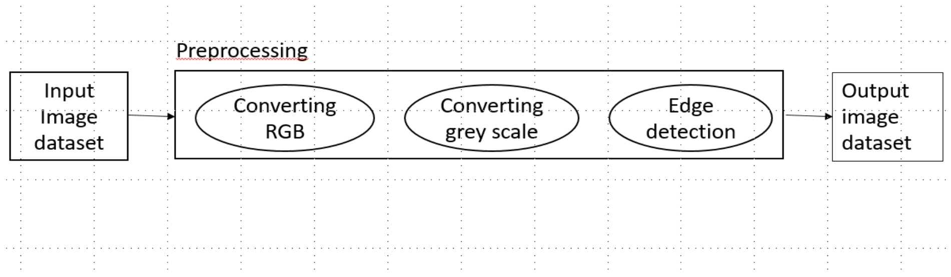
Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Digital image processing is the use of computer algorithms to create, process, communicate, and display digital images. Digital image processing algorithms can be used to: Convert signals from an image sensor into digital images. Improve clarity, and remove noise and other artifacts.

Step 2: Feature Extraction

In machine learning, pattern recognition and in image processing, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction.

Block Diagram



Algorithm

Algorithm 1 Detection of Horizon In Maritime Images- Preprocessing
Input : Upload Image Dataset
Output : Images with horizon detected

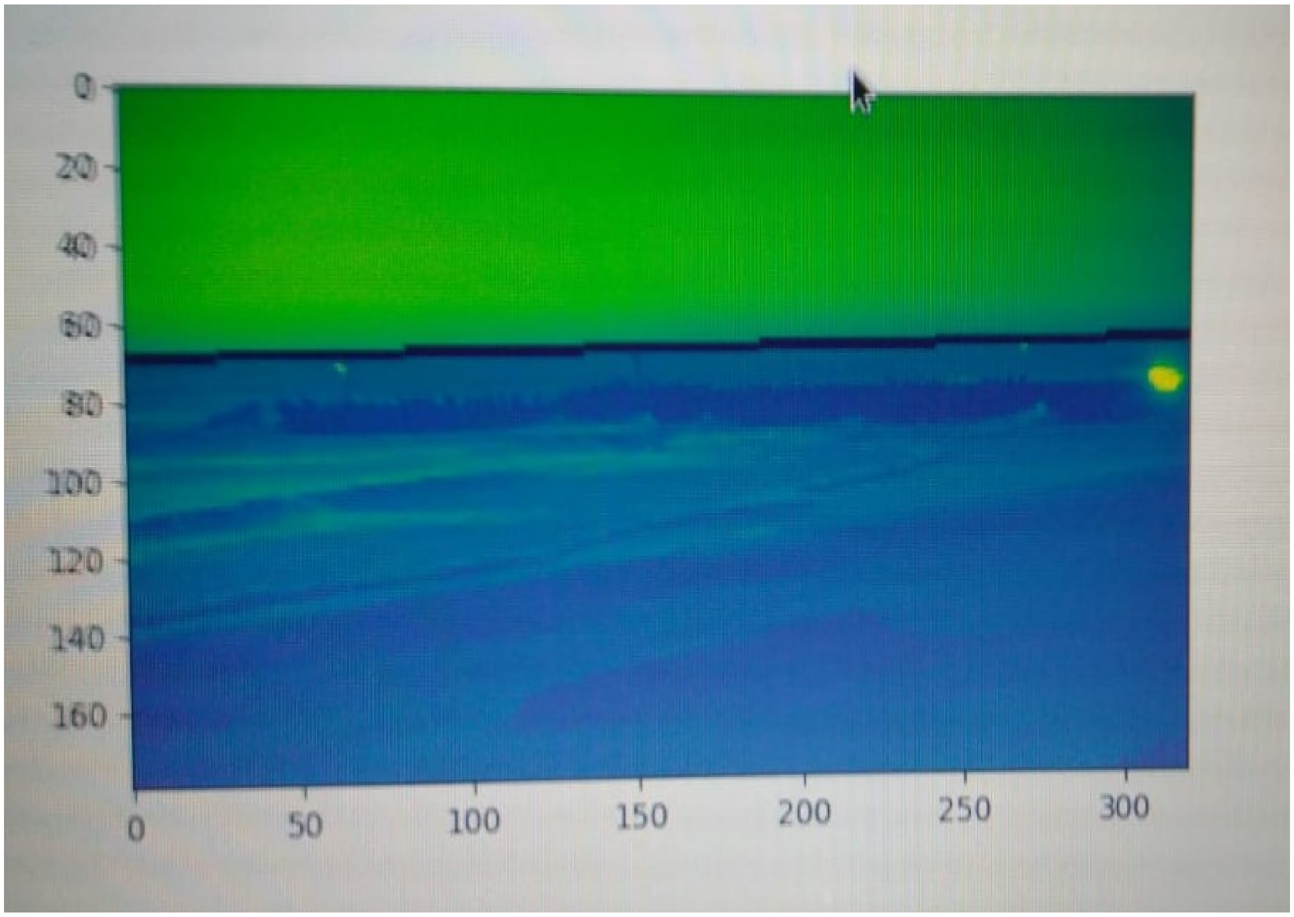
```
Step1 : Start
Step2 : Input the image dataset.
Step3 : Edge is detected using canny edge detection Algorithm.
Step4 : Images with Edge detected detected
Step5 : stop
```

Edge detection algorithms operate on the premise that each pixel in a grayscale digital image has a First derivative, with regard to the change in intensity at that point, if a significant change occurs at a given pixel in the image, then a black pixel is placed in the binary image, otherwise, a white pixel is placed there instead.In general, the gradient is compared at each pixel that gives the degree of change at each point in the image.The question basically amounts to how much change in the intensity should be required in order to constitute a feature point in the binary image.And usually a predefined threshold value T is used to classified edge points.To find the accuracy location of an edge, a Second derivative is often used to find the point that corresponds to the local maximum and minimum in the first derivative.This is often referred to as a Zero Crossing because it is the point at which the second derivative equals to zero, but its left and right neighbors are non-zero and have opposite signs.

5.1.2 Horizon detection using Houghline transform

The Hough Transform (HT) is a robust method for finding lines in images that was developed by Paul Hough.For each line L, there is a unique line perpendicular to L which passes through the origin. L?has a unique distance and angle from the horizontal axis of the image. This angle and distance define a point in the parameter space, sometimes known as Hough space.A point in image space has an infinite number of lines that could pass through it, each with a unique distance and angle.This set of lines corresponds to a sinusoidal function in parameter space. Two points on a line in image space correspond to two sinusoids which cross at a point in parameter space.That point in parameter space corresponds to that line in image space, and all sinusoids corresponding to points on that line will pass through that point.

Horizon Line



Chapter 6

ISSUES FACED AND REMEDIES TAKEN

- **Issue 1 :**Initially the project is started using dataset of Maritime Images.That is not at all clear for fredicting the Horizon.
- **Remedy :** change the dataset and select the clear images.
- **Issue 2 :** There was as an issue while running the code , showing a NameError.
- **Remedy :** Installed required libraries.
- **Issue 3:** It was difficult to find out edges and pixels in the images
- **Remedy :** Use the edge detection algorithm to overcome that.

Chapter 7

RESULT ANALYSIS

7.1 Result

Detection of horizon in an image is an important part of many image related applications such as detecting ships on the horizon, flight control, and port security. Most of the existing solutions for the problem only use image processing methods to identify a horizon line in an image. This results in good accuracy for many cases and is fast in computation. Single image horizon line estimation is one of the most fundamental geometric problems in computer vision.

However, for some images with difficult environmental conditions like a foggy or cloudy sky these image processing methods are inherently inaccurate in identifying the correct horizon. This investigates how to detect the horizon line in a set of images using a machine learning approach. When executing the python code the Horizon Detection of Maritime Images have detected .Horizon have seperated by Using a single line. The line whch seperate both sky and the non-sky.

Chapter 8

CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

Detecting a horizon in an image is an important part of many image related applications such as detecting ships on the horizon, flight control, and port security. Most of the existing solutions for the problem only use image processing methods to identify a horizon line in an image. This results in good accuracy for many cases and is fast in computation.

However, for some images with difficult environmental conditions like a foggy or cloudy sky these image processing methods are inherently inaccurate in identifying the correct horizon. b This investigates how to detect the horizon line in a set of images using a machine learning approach.

8.2 Future Scope

In future, to find out the object in the horizon. That means if an object if it is a flight or a ship whatever that consists in sea or sky. That is the future scope .To find out the object in which Horizon.

Chapter 9

APPENDIX

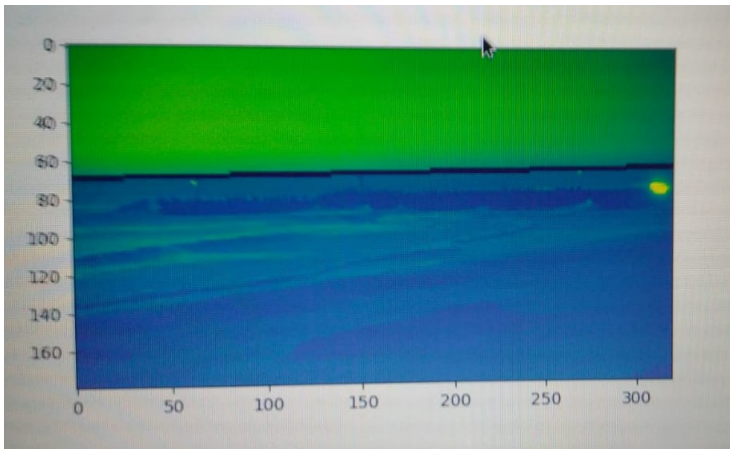
9.1 Source Code

Horizon.py

```
from __future__ import division

import numpy as np
import cv2
from matplotlib import pyplot as plt
import os
import time
import math

current_dir = os.getcwd()
files = [x for x in os.listdir(current_dir) if x.endswith('.jpeg')]


def get_plane_indicator_coordinates2(img,angle,distance):

    x1 = int(round(max(0,
                (img.shape[1]/2)-((img.shape[0]/2)/math.tan(abs(angle)*np.pi/180))))
    x2 = img.shape[1] - x1

    if i > 0:
        y1 = int(round(min(img.shape[0],
            (img.shape[0]/2) + ((img.shape[1]/2) *
                np.tan(angle*np.pi/180)))))
    else:
        y1 = int(round(max(0,
            (img.shape[0]/2) + ((img.shape[1]/2) *
                np.tan(angle*np.pi/180)))))
    y2 = img.shape[0] - y1

    return [(x1,y1),(x1,y2)]

def get_plane_indicator_coord(img,angle,dist_as_perc):
    '''
    img: image opened in cv2
    angle: angle of plane indicator in degrees
    dist_as_perc: pitch, as percent of 100 in which 0.50 is the
        origin
    '''

    heading = 90 + angle
    heading_from_hor =
        min(math.radians(180-heading),math.radians(heading))

    radius = int(math.ceil(math.sqrt((img.shape[1]/2)**2 +
        (img.shape[0]/2)**2)))

    x0 = img.shape[1]/2
    y0 = img.shape[0]/2
```

```
x1 = (img.shape[1]/2) - radius
x2 = (img.shape[1]/2) + radius

y1 = (img.shape[0]/2)
y2 = (img.shape[0]/2)

x1_n = x0 + math.cos(math.radians(angle)) * (x1 - x0) -
    math.sin(math.radians(angle)) * (y1 - y0)
y1_n = y0 + math.sin(math.radians(angle)) * (x2 - x0) +
    math.cos(math.radians(angle)) * (y2 - y0)

x2_n = x0 + math.cos(math.radians(angle)) * (x2 - x0) -
    math.sin(math.radians(angle)) * (y2 - y0)
y2_n = y0 + math.sin(math.radians(angle)) * (x1 - x0) +
    math.cos(math.radians(angle)) * (y1 - y0)

if heading_from_hor < math.atan(img.shape[0]/img.shape[1]):
    avail_dist =
        int(img.shape[1]/math.cos(math.radians(heading)))
else: #heading_from_hor >= np.arctan(img.shape[0]/img.shape[1])
    avail_dist =
        int(img.shape[0]/math.sin(math.radians(heading)))

origin = avail_dist / 2
heading_transform = (dist_as_perc * avail_dist) - origin

x_transform = heading_transform * math.cos(math.radians(heading))
y_transform = heading_transform * math.sin(math.radians(heading))

return [(int(x1_n+x_transform),
         int(y1_n+y_transform)),
            (int(x2_n+x_transform),
             int(y2_n+y_transform))]
def show_horizon(img_file):
    img = cv2.imread(img_file,0)
    img = cv2.resize(img,dsize=None,fx=0.25,fy=0.25)
    img_ann = img

    edges = cv2.Canny(img, 40, 60, apertureSize = 3)
    lines = cv2.HoughLines(edges, 1, np.pi/180, 80)
    line_info =
        np.array([0,720,1280,720,10000,10000,10000],dtype=int)

    for i in range(len(lines)):
        rho,theta = lines[i][0]
        a = np.cos(theta)
        b = np.sin(theta)
        angle_from_horizon = np.pi/2 - theta

        if np.abs(angle_from_horizon) < np.radians(45):
            c = np.tan(angle_from_horizon)

            x0 = a*rho
            y0 = b*rho

            x1 = 0
            y1 = int(y0 + (x0*c))

            x2 = img.shape[1]
            y2 = int(y1 - (x2*c))

            global_sky_mat = np.empty(0,dtype=np.uint8)
            global_sea_mat = np.empty(0,dtype=np.uint8)
            local_sky_mat = np.empty(0,dtype=np.uint8)
            local_sea_mat = np.empty(0,dtype=np.uint8)

            for x_coor in range(0,x2,2):
                y_d = int(y1 - (x_coor * c))

                global_sky_mat = np.append(global_sky_mat,
                    img[0:y_d,x_coor] )
                #global_sea_mat = np.append(global_sea_mat,
                    img[y_d:img.shape[0], x_coor])

                local_sky_mat = np.append(local_sky_mat,
                    img[(y_d-10):y_d, x_coor] )
                local_sea_mat = np.append(local_sea_mat,
                    img[y_d:(y_d+10), x_coor] )

            global_sky_var = int(np.var(global_sky_mat))
            #global_sea_var = int(np.var(global_sea_mat)
            local_sky_mean = int(np.mean(local_sky_mat))
            local_sea_mean = int(np.mean(local_sea_mat))

            stats = np.array([x1,y1,x2,y2,
                              global_sky_var,
                              local_sky_mean,
                              local_sea_mean])

            #if global_sky_var <= global_sea_var:
            line_info = np.vstack((line_info,stats))

    start = time.time()
    print('finding best line...')
    # minimize intraclass variance of sky and sea
    # maximize interclass mean of sky and sea
    #(line_info[:,4]+line_info[:,5])/

    #best_line = line_info[np.argmin(
    #
        (line_info[:,4])**2/np.abs(line_info[:,6]-line_info[:,7]))]

    #best_line = line_info[np.argmin(
    #               (line_info[:,4])**2 +
        line_info[:,5])/np.abs(line_info[:,6]-line_info[:,7]))]

    #best_line = line_info[np.argmin(line_info[:,4]+line_info[:,5])]

    best_line = line_info[np.argmin(
                   (line_info[:,4])/np.abs(line_info[:,5]-line_info[:,6]))]

    end = time.time()
    cv2.line(img_ann,(best_line[0],best_line[1]),(best_line[2],best_line[3]),(0,0,255),2)

    print(img_file)
    print(best_line)
    print(len(lines))
    plt.imshow(img_ann)
    plt.show()

for f in files:
    show_horizon(f)
```

9.2 Screenshot



Chapter 10

REFERENCES

1 .1. S.M. Ettinger, M.C. Nechyba, P.G. Ifju, M. Waszak, "Vision Guided Flight Stability and Control for Micro Air Vehicles", Advanced Robotics, vol. 17, no. 7, pp. 617-40, 2003.

2 . T.G. McGee, R. Sengupta, K. Hedrick, "Obstacle Detection for Small Autonomous Aircraft Using Sky Segmentation", Proceedings of the IEEE International Conference on Robotics Automation, 2005.

3 . S. Todorivic, M. Nechyba, "Sky/ground modelling for autonomous MA V flight", Proc. of IEEE International Conf. on Robotics and Automation, 2003.

4 . R.C. Gonzalez, R.E. Woods, L.E. Steven, "Digital Image Processing using Matlab" in , Pearson Prentice Hall, 2004.