Exercise 4

Author: Daniel Lindberg

1)Richard O. Duba and Peter E. Hart tackled a problem involving detecting lines and curves in images. The original method to detect lines was to be able to detect presence of groups of collinear points which was in turn a bit computationally expensive. Originally detecting a straight line, they would generate a slope, however it posed a problem on vertical lines which typically have their slope hard to define in and x,y coordinate plane. Duba and Hart proposed the use of the a “normal parameterization”, that specifies a line by the angle θ. The equation of a line now becomes xcos(θ) + y sin (θ) = p. Where p is the algebraic distance from the origin. They would then be able to associate a p and a θ to get the angle and the distance to a set straight line. Given a single point in the plane, then the set of all straight lines going through that point corresponds to a sinusoidal curve in the (p,θ) plane. A set of two or more points that form a straight line will produce sinusoids which cross at the (r,θ) for that line. This would change the problem into finding concurrent curves instead of collinear points. There are some limitations of the transform approach. Results may be sensitive to the quantization of both θ and p, the more precise gives better resolution but increases how computationally expensive it is. In addition this technique finds collinear points without regard to how continuous they are, the presence of unrelated points in the digital images may produce distortion. Finally this method can also apply to detecting circular figures in an image, which involves taking the normal parameterization along with the transform to detect the points in a circle appearing on a circular cone in a three dimensional parameter space. This transform methods can apply to any curves that we define as long as they pick a convenient parameterization for that said curve. (https://www.cse.unr.edu/~bebis/CS474/Handouts/HoughTransformPaper.pdf)

2) You can find my images located in my Step2 Folder within my zipped file report. They are denoted as : Alaska-air.jpg , alaska-air-hough-line.png and alaska-air-hough-line-figure.png.

3)
I adapted the C++ code from Skeletal.cpp to python. In my Step3 folder you can find my code titled “skeleton.py”. To run it simply type “python skeleton.py”. This will generate Frames for 100 seconds and have the title as “FrameXXXX.png” . Where XXXX is a 4 digit of the frame number. Then how I compiled it into my mpeg4 video I ran the following command in Linux: ffmpeg -i Frame%04d.png -c:v mpeg4 Skeleton.mp4

My video then in turn is titled Skeleton.mp4

4) Just like in the previous step I wrote my code in Python. You can find it in the Step4 folder. You can find it titled “Thinning.py”. This will generate Frames for 100 seconds and have the title

as “FrameXXXX.png” . Where XXXX is a 4 digit of the frame number. Then how I compiled it into my mpeg4 video I ran the following command in Linux: ffmpeg -i Frame%04d.png -c:v mpeg4 BottomUpSkeleton.mp4

I believe that the E.R. Davis’s function may be better at producing a “figure skeleton”. However it was slower on my system then the top down approach based off of the timing sample I took from the two codes running. The first one takes each frame and gets the iterations to the very bottom most sample. E.R. Davis describes the process of “Thinning” . They describe it as stripping away the outermost layers of a figure until only the connected width skeleton remains. I believe this solution that I have produced has a number of skeletons in the background that it ends up drawing. I could probably fix this issue by changing the threshold value before I sent it over to my “Thinning” algorithm. According to E.R. Davis in certain parts of different shapes the skeleton segments are part of parabolas rather than straight lines. Which may result in a detailed shape of the skeleton may not exactly be expected. (E.R. Davis Computer & Machine Vision Theory Algorithms Practicalities fourth edition 2012)

5) Image matching is a fundamental concept in many problems in computer vision, which involves scene recognition, 3D structure formation from imaging, motion tracking.. Etc. There are certain features which help with matching aspects of an image or scene. The features must not be effected by image scaling, rotation, illumination and 3D camera viewpoint. Some of these features should be able to be extracted at a certain locations that pass an initial test. It does a scale-space extrema detection which searches over all scales and image locations to identify potential interest points that are not affected by scale and variant. It uses keypoint localization , which at each candidate location has a detailed model to determine the location and scale, then keypoints are selected based on the measures of how stable or how they are moving in between a scene. Orientation assignment where they review each keypoint location to see if the object has been transformed relative to the assigned orientation, scale and location for each feature. Finally keypoint descriptors where local image gradients are measured around a keypoint to allow for significant levels of shape distortion and change in illumination. These approaches have combined into a method called the Scale Invariant Feature Transformation. This method will allow a number of features to be determined , even in cluttered backgrounds which requires 4 features to match from each object for identification. SIFT features are obtained via reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to the previous within the database and finding matching features based on Euclidean distance of their features vectors. Sometimes the background features that are determines often times do not match to anything within the database giving rise to false matches. Each cluster of 3 or more features that agree on an object and its location are then used for further verification. This is pivotal use of technology since SIFT can robustly identify objects even among clutter and is not affected by scaling, orientation, illumination changes and distortion.

6) I tried to compile the code in the exercise but is not compatible with opencv version 3.2. So I wrote custom code in python which essentially does the same drawings. If you go into my Step6 folder you can find the python translation code called sift.py to run it simply run “python sift.py”. I ran the code using my First.jpg and my Second.jpg. The result image is located at: Frame_Differencing.png

7) The code that was provided had a number of not only spelling errors but also library errors. I wrote a simple python script that utilizes the stereo_match functionality. My python script is titled stereo_match.py and can be found in the Step7 folder. I have a sample output image of the script named: Sample_Python_Finish.png. This image shows the stereo_match on just the snapshot left. To run the code simply type “python stereo_match.py”

How this is determined is by determining the distance between two cameras and the focal length of the camera. So you can determine the depth of a point in a scene is inversely proportional to the difference in distance of corresponding image points and their camera centers. So you can begin to get the depth of all pixels within an image.

You generate a disparity map , and the result can contain noise. You can adjust the number of disparities and block size to get better results.