Exercise #4 – Continuous Transformations for Scene Understanding

DUE: AS INDICATED on Canvas

Please thoroughly read posted pages from Computer and Machine Vision by E.R. Davies on Canvas and notes on Segmentation; please also note the scanned special handout from E.R. Davies Computer and Machine Vision Chapter 9 on methods to skeletonize an image (machine vision process) uploaded on Canvas. The goal of this lab is to introduce concepts and theory related to continuous scene segmentation, basic shape recognition (with skeletal thinning), depth estimation, and behavioral analysis, which feed into recognition methods. It includes both top-down OpenCV application development and understanding as well as bottom-up algorithm implementation and understanding. Please see Code-Examples and Lab4-Examples. To dig deeper into machine learning research examples, this resource can be helpful - https://paperswithcode.com/area/computer-vision, e.g., on a specific topic such as https://paperswithcode.com/task/pedestrian-detection especially as you think about your final projects).

Exercise #4 Goals and Objectives:

1) [10 points] Read the paper Use of the Hough Transformation to Detect Lines and Curves in Pictures, Richard Duda & Peter Hart (also available on Canvas) and summarize the papers key points and contributions to computer vision.

2) [20 points] Applying Hough Lines to AV/ADAS: Build and run the basic Hough lines example provided by OpenCV and adapt it for use with a camera or video source with the ability to specify the source as camera or video on the command line. Determine the best method (***HoughLines***, ***HoughLinesP***) and parameters to use to find the edges of a roadway from the following video – Driving Challenge Set 1 or Driving Challenge Set 2 (***front facing*** is the best for lane following tests). Paste a transformed image showing your best detection of roadway edges and lanes in your report.

3) [20 points] Using a top-down OpenCV approach, adapt the example code for OpenCV version used in class based on examples found in capture_transformer to use the skeletal.cpp transform on continuous frames (like captureskel.cpp) from your camera, but use the much simpler approach of simpler-capture rather than V4L2 camera capture. Gesture in front of your camera and see if you can get a reasonable continuous skeletal transform of your arm and hand, holding up 1, 2, 3, 4, or 5 fingers to see if you could distinguish each and count fingers held up. Record example frames (up to 3000 for 100 seconds of video – JPEG frames are fine) and encode your results to an MPEG video. Upload the modified code with your report.

4) [10 points] Read the paper Distinctive Image Features from Scale-Invariant Keypoints, by David Lowe (also available at csci612/code/siftDemoV4/) and summarize the papers key points and contributions to computer vision. Download the sample code here - https://www.cs.ubc.ca/~lowe/keypoints/, siftDemoV4.zip) and run the demos and show that you have done this by incorporating a screen shot of correspondence between book and scene and providing a brief explanation – create keys in Windows or MATLAB (also found here – book.key & scene.key). The Pyramidal transform (https://docs.opencv.org/4.1.1/d4/d1f/tutorial_pyramids.html) is a key part of SIFT, download PyrUpDown code and describe what happens if you zoom in and then zoom out at the default level of zoom several times, returning to the original zoom level and explain what happens.

Option #1 – Deeper dive on Methods of Detection using HOG with standard OpenCV 4.1.1 code (no need to build OpenCV from sources, and no need to download and build contrib and extra code samples). ***This is the simplest and recommended detection/recognition exercise.***

5) [20 points] Build and run the basic OpenCV people detector example. (https://docs.opencv.org/4.1.1/df/d54/samples_2cpp_2peopledetect_8cpp-example.html) and test it to see how well it works on to detect people. You can use your camera and/or you can use a challenge set such as – Pedestrian Challenge Set Videos from Caltech Vision Datasets (described here). Report your results showing images where you were able to detect people or other objects of interest. Try the example on simple single images as well.

6) [20 points] Study the references the opencv.org provides for HOG descriptors (https://docs.opencv.org/4.1.1/d5/d33/structcv_1_1HOGDescriptor.html) including: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients, and https://learnopencv.com/histogram-of-oriented-gradients/. Could you adapt HOG using training (e.g., train_HOG.cpp and hog.cpp) to either improve pedestrian detection (for Colorado, Arizona, or Caltech challenge data – pedestrians can be found in most of this data) OR detect other objects of interest such as stop signs (https://unsplash.com/s/photos/stopsign, and https://www.kaggle.com/datasets/andrewmvd/road-sign-detection) or other signs? Note that you can find databases of images from Imagenet or from our challenge set data for CSCI 612 and other public ML datasets (csci612/media/ and CSCI-612-ACV). Download and run the opencv.org HOG training code examples and attempt to train and use your own HOG descriptors for a new object such as stop signs OR to improve pedestrian detection compared to pre-trained people detector. Show your results by example.

Option #2 – Deeper dive on gesture recognition methods comparing OpenCV with MediaPipe. Compare methods to segment fingers (with test to determine how many fingers are showing – counting them) using OpenCV skeletal transforms (as done in #3) or OpenCV

contours (see example here – Hand and Finger Detection App, Find & Draw Contours, Paper on Hand gesture recognition) with MediaPipe method (gesture recognizer).

5) [20 points] Build and run your skeletal gesture recognizer (assuming you got it to work) or use the OpenCV contour method from CSCI 612 prior project work and test and analyze the performance with a confusion matrix (Wikipedia: Confusion Matrix – where {one, two, three, four, five, none} X {one, two, three, four, five, none}). Note that a confusion matrix that is perfect will only have counts on the diagonal (all ones are recognized at ones and so forth up to fives). For off diagonal counts, explain why you think the top 3 off diagonals are being confused.

6) [20 points] Repeat the confusion matrix analysis above now using the MediaPipe gesture recognizer. Report results as a confusion matrix again. Which works better? MediaPipe or OpenCV method you chose? Why?

Option #3 – Deeper dive using SIFT with OpenCV contrib and extra code (git clone https://github.com/opencv/opencv and git clone https://github.com/opencv/opencv_contrib) after installing all of OpenCV 4.x ground up from sources on your Jetson Nano. This is non-trivial, but is quite possible, so ***take Option #3 only if you are really interested in passive stereo beyond the basic introduction*** based on David Lowe's example code. More current versions of OpenCV, e.g., 4.12 at the time of writing this lab procedure may now have SIFT in the main library, so double check before you re-build all of OpenCV (time consuming and error prone).

5) [20 points] Build and run the basic OpenCV extra/contrib code keypoint comparison detector code provided in sift as detector_extractor_matcher.cpp and paste a transformed image of your choice into your report. Note that you will need to update for the latest OpenCV 4.x and the latest contrib/extra OpenCV code. It should provide a comparison like this between two images:

Take two snapshots of the same scene (as was done above) from a left/right offset and run the detector comparison code and paste the result into your report.

6) [20 points] Build and run the contrib/extra code based disparity depth estimator example in example_stereo using two cameras (or one camera, moving it left or right) and describe how it works based on your reading and understanding of the code and just in terms of performance and accuracy. Paste in an example left-eye, right-eye, depth map set of three images that correspond which you captured with your cameras. Are keypoints used for passive depth estimation, and if not, could they be?

Option #4 – Deeper dive on Face Recognition with OpenCV contrib and extra code (git clone https://github.com/opencv/opencv and git clone https://github.com/opencv/opencv_contrib) after installing all of OpenCV 4.x ground up from sources on your Jetson Nano. This is non-trivial, but is quite possible, so take Option #4 only if you are really interested in face recognition. More current versions of OpenCV, e.g., 4.12 at the time of writing this lab procedure may now have support in the main library, so double check before you re-build all of OpenCV (time consuming and error prone).

5) [20 points] Build and run the basic OpenCV face recognition example. (https://docs.opencv.org/4.1.1/da/d60/tutorial_face_main.html) and test it to see how well it works using one of the example databases (e.g., ATT). Report your results showing images where you were able to detect people or other objects of interest. Try the example on simple single images as well.

6) [20 points] Create your own face database and training and create an OpenCV application that can recognize your face and distinguish from other faces, using captured images or with your camera.

Option #5 – Deeper dive on CUDA with OpenCV contrib and extra code – there are two options for developing ACV CUDA code and the best is to implement the algorithm yourself and create a CUDA kernel for it – the OpenCV libraries that install with the NVIDIA SDK typically already make use of the GP-GPU to a degree, but you may also be able to get the best performance by re-building the latest OpenCV (4.12 at the time of writing this lab – 4.11 CUDA Docs) along with the contrib code – however, you can typically install CUDA with "sudo apt-get install cuda" for development of your own custom implementation (Tutorial-CUDA-and-Multi-Core.pdf) and don't need to rebuild all of OpenCV. You can find and install "jtop" using https://github.com/rbonghi/jetson_stats.

Think twice before rebuilding OpenCV libraries: ***Re-building all of the existing OpenCV libraries for new or additional CUDA support is a non-trivial exercise. Often the pay-off for rebuilding is low unless you need the latest cuDNN for machine learning models or very specific feature not available in the default SDK OpenCV (check before you rebuild).***

I recommend just writing your own CUDA kernel based on first principles for your transformation (convolution) or visual cue of interest for this rather than relying on the OpenCV library CUDA support.

5) [20 points] Build and run the basic NVIDIA CUDA example code and pick 3 applications to evaluate from the NVIDIA samples (https://github.com/NVIDIA/cuda-samples - note that you may have to update the Makefile as I have done for the transpose example, or deviceQuery and my 2 favorites – smoke & nbody). Also note that I have debugged and built most of the CUDA samples for the A100 (cuda-samples-a100.tar.gz) other than graphical output examples, but the goal here is to understand the value of CUDA on your Jetson Nano, which also has graphics. Report your results showing CPU loading with "htop" and GP-GPU loading with "jtop." Do any of the 3 applications you have selected saturate the GP-GPU and if not, why not?

6) [20 points] Create your own CUDA accelerated application for your centroid tracker from lab #3 or any other image transform you would like other than sharpen (e.g., Sobel, Laplace, DCT, etc.) [note that these examples I provide from previous labs may help - sharpenCUDA/ stand-alone, or as an 8th example - 8_ACV_CUDA_Examples/CUDA_vs_Mulit-Core-compare/]. Show that your CUDA code works and that it is running on the GP-GPU by showing loading with "jtop."

Upload all images as PPM, PGM, PNG, or JPG and video as encoded MPEG-4 at a reasonable bitrate and quality.

Overall, provide a well-documented professional report of your findings, output, and tests so that it is easy for a colleague (or instructor) to understand what you have done. Include any C/C++ source code you write (or modify) and Makefiles needed to build your code and make sure your code is well commented, documented and follows coding style guidelines. I will look at your report first, so it must be well written and clearly address each problem providing clear and concise responses to receive credit.

Upload all code and your report completed using MS Word or as a PDF to Canvas and include all source code (ideally example output should be integrated into the report directly, but if not, clearly label in the report and by filename if test and example output is not pasted directly into the report). ***Your code must include a Makefile so I can build your solution on an embedded Linux system (R-Pi 3b+ or Jetson). Please zip or tar.gz your solution with your first and last name embedded in the directory name and/or provide a GitHub public or private repository link. Any code that you present as your own that is "re-used" and not cited with the original source is plagiarism. So, be sure to cite code you did not author and be sure you can explain it***

in good detail if you do re-use, you must provide a proper citation and prove that you understand the code you are using.

Grading Rubric

[10 points] Read and summarize the main points of Duda and Hart paper on Hough transform.

Problem	Points Possible	Score	Comments
3 main points of the paper	6		
Summary overall	4		
TOTAL	10		

[20 points] Build, run and tune Hough linear example and adapt it for video or camera sources.

Problem	Points Possible	Score	Comments
Build, run, adapt, and tune for test and demonstration with camera or video	10		
Application of Hough lines to challenge set video for lane detection	10		
TOTAL	20		

[20 points] Continuous skeletal transform of you hand

Problem	Points Possible	Score	Comments
OpenCV code, build, run, test for continuous skeletal transformation	10		
Recording of 100 seconds of video showing that your continuous skeletal transformation works and tested with # of fingers held up	10		
TOTAL	20		

[10 points] Read and summarize the main points of David Lowe paper on SIFT.

Problem	Points Possible	Score	Comments
3 main points of the paper and demonstration of code examples (SIFT and PyrUpDown)	6		
Summary overall	4		
TOTAL	10		

[20 points] Build and option starter code and examples – Option #1, #2, #3, #4 or #5.

Problem	Points Possible	Score	Comments
Build and run completion evidence	10		
Testing to show basic performance	10		
TOTAL	20		

[20 points] Adapt and extend starter code and examples – Option #1, #2, #3, #4 or #5.

Problem	Points Possible	Score	Comments
Build and run to demonstrate your prototype application	10		
Testing to show basic performance or your application	10		
TOTAL	20		

Report file MUST be separate from the ZIP file with code and other supporting materials.

Rubric for Scoring for scale 0…10

Score	Description of reporting and code quality
0	No answer, no work done
1	Attempted and some work provided, incomplete, does not build, no Makefile
2	Attempted and partial work provided, but unclear, Makefile, but builds and runs with errors
3	Attempted and some work provided, but unclear, build warnings, runs with no apparent error, but not correct or does not terminate
4	Attempted and more work provided, but unclear, build warnings, runs with no apparent error, but not correct or does not terminate
5	Attempted and most work provided, but unclear, build warnings, runs with no apparent error, but not correct or does not terminate
6	Complete answer, but does not answer question well and code build and run has warnings and does not provide expected results
7	Complete, mostly correct, average answer to questions, with code that builds and runs with average code quality and overall answer clarity
8	Good, easy to understand and clear answer to questions, with easy-to-read code that builds and runs with no warnings (or errors), completes without error, and provides a credible result
9	Great, easy to understand and insightful answer to questions, with easy-to-read code that builds and runs cleanly, completes without error, and provides an excellent result
10	Most complete and correct - best answer and code given in the current class