Assignment 5 - Extended Lab Proposal and Exploratory Coding
Author: Daniel Lindberg

1) In the article “Stanley: The Robot that Won the DARPA Grand Challenge” , Journal of Field Robotics 23(9), 661-692 (2006), written by Sebastian Thrun he details out why the Stanely vehicle required machine vision for the driving challenge. The car that was developed for this challenge and eventually won used multiple sensors on the car to map the terrain of the course. They used the laser mapper to detect obstacles and terrain of up to 22 meters away from the car, this became an obstacle when the car was reaching speeds faster than 25 mph where they would have to get a grasp of the terrain further than the 22 meters capabilities of the laser sensors. They used a color camera which in turn was able to classify parts of the image into driveable and non driveable regions. They had limitations to their algorithm since some factors were not easily measure and would rapidly charge, such as the surface material of the road, lighting conditions, dust on the lens . They were trying to detect driveable surface past the laser sensor range, so they would take the data from the laser analysis and put it into the camera image. It would then determine a quadrilateral ahead of the robot in the laser map. They would use a mixture of Gaussian blurs that would model the color of the terrain to train an algorithm to determine driveable terrain. Then every new image coming into the car would use the quadrilateral gaussian values using the learning algorithm to allow for fast adaptation to the road. Sometimes the terrain would change color and texture so the learning algorithm had to change on the fly, they would have to adapt this learning algorithm with new images to adapt to the new surface color. Under slow changing lighting the system would adapt slowly to the road surface. When this algorithm could not detect drivable road within a 40 meter range it slowed down to 25 miles per hour for safter navigation.

   It is an interesting debate to see if the Stanley vehicle could have just used a more sophisticated machine vision system. Since the laser sensor receives data at 75 Hz and from each of the cameras at approximately 12 Hz. This would mean that the data received from the lasers was at a must faster rate. Since in the previous description we talked about how the laser sensors would give the terrain and drivable distance to the images within a 22 meter range, then camera was in charge of all distances greater than that 22 meter range. Part of me believes that these laser sensors were necessary because they were faster read, and then the data from the lasers would be put into the images to be processed. I think the cameras may have not been able to receive data at a fast enough rate or not enough speed to process at the time of the creation of the Stanley vehicle. Now that we are in the future and other such automated cars have come out we have seen machine vision used more in some of these instances. Tesla’s algorithm for driving automation has not publicly been disclosed how the driving technology works, but some people believe it uses data from radar and ultrasound sensors. Even Uber and Baidu’s self driving cars used Lidar. I just can’t imagine a system that can develop a 3D model space fast enough with current capabilities , but if data could be extracted from cameras and processed up to a certain speed I would like to believe that you could use the previous lab's stereo match system to detect motion. We have done in the class the sobel and the canny methods to detect gradients/terrain, so that has some of the features of self driving cars. I think if we had a magical way to extract camera images at incredibly fast speeds and process them almost instantaneously and there were no factors that would affect the camera such as dirt obscuring it I believe it would be possible.

2) I did the Pedestrian Detection section of the code, which is selection B. You can find my code in the Step2 folder. The code is titled TrackOverlay.py. It takes 4 arguments as specified in the Exercise 5 requirements document. The first parameter is if you want to show the boxes being drawn in the display. The second parameter is the video input file we want to do the pedestrian detection on. The third argument is the title at the top of the frame of the displayed image. The final argument is the store argument which tells the name of the new video file that will be produced with the pedestrian detection frames.

   A sample call to my program would be: "python TrackOverlay.py True people.mp4 ShowName NewVideo"

   It will then display the coordinates of the rectangles of the found image within the people.csv file. The video will be the name of the video parameter passed with the file extension being .avi.

   How the code works: It takes in the HogDescriptor Opencv Class. This is the histogram of gradients class, and it involves feature descriptors. A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. We are going to use this feature descriptor to detect the features in a pedestrian, typically a head, two arms, a torso and a pair of legs. OpenCV has a default person detector built in using a linear SVM model. SVM stands for support vector machine. These support vector machines take in a learned training set to detect these features within a human. We invoke this default person detector by using the: cv2.HOGDescriptor_getDefaultPeopleDetector(). They take positive samples from the training set to the SVM , in addition they add in negative samples into the training set which does not have the feature. They will tell the trainer that the 'positive' images or the ones with pedestrians within it are positive and the negative photos do not have negative photos. Then they look at the trainer and see if there is any false-positives, and sort them by confidence. Then they retrain using some of these hard-negative images.

   In my code I detected a number of "pedestrian" feature examples. So I added in a second function that would use the non_max_suppresion so that if there are a number of rectangles within a certain section of the image, it would then draw a larger rectangle to encompass all of the related frames. Therefore no rectangles drawn within rectangles.

3) I would like to express more interest in doing pedestrian recognition. I thought it may be interesting to apply the methods I did in step 2 towards UAV footage, my algorithm needs some work and I think I could improve upon it. This would be a challenging proposal. My goal is to take potentially some of the UAV footage from Professor Siewert or some other UAV footage with people in the background and find a way that implements an: accurate support vector machine, methodology to effect in scaling of the image, the processing power to take the frames (depending on the image size) and process them through my support vector machine, read the video frames at a fast enough I/O rate.

I think 30 frames per second on a 640x480 px images utilizing 1 core and no GPU having a jitter of plus or minus 2. I plan to utilize the methodology I describe in Step 2 of this exercise, also listed below:

How the code works: It takes in the HogDescriptor Opencv Class. This is the histogram of gradients class, and it involves feature descriptors. A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. We are going to use this feature descriptor to detect the features in a pedestrian, typically a head, two arms, a torso and a pair of legs. OpenCV has a default person detector built in using a linear SVM model. SVM stands for support vector machine. These support vector machines take in a learned training set to detect these features within a human. We invoke this default person detector by using the: cv2.HOGDescriptor_getDefaultPeopleDetector(). They take positive samples from the training set to the SVM , in addition they add in negative samples into the training set which does not have the feature. They will tell the trainer that the ‘positive’ images or the ones with pedestrians within it are positive and the negative photos do not have negative photos. Then they look at the trainer and see if there is any false-positives, and sort them by confidence. Then they retrain using some of these hard-negative images.

In my code I detected a number of “pedestrian” feature examples. So I added in a second function that would use the non_max_suppresion so that if there are a number of rectangles within a certain section of the image, it would then draw a larger rectangle to encompass all of the related frames. Therefore no rectangles drawn within rectangles.

So we detailed out a frame rate goal, I would like to hopefully recognize every person in every frame and get the coordinates of their bounding rectangle, I would like for this algorithm to also scale. Right now my code is written to take in features of people in the foreground so having it scale with how far people may be away in the image is something I would believe is a realistic goal.

Success would be having: 30 fps on 640x480 images utilizing my 1 Core no GPU. Recognition of pedestrians in every scene of the video footage and finally having this code work across multiple datasets and not just one training set and video footage.