Embedded Machine Vision and Intelligent Automation

Exercise 5

Sarthak Jain

July 19, 2019

Performed on Jetson Nano

Performing Final Project with Siddhant Jajoo

Question 1

1. Use of Machine Vision: The vehicle Stanley used laser mapping as the basis for short and medium-range obstacle detection. However, the maximum effective distance up to which lasers could detect obstacles was 22m. This range was insufficient for the maximum speed that Stanley had to achieve, and so for longer-range obstacle detection, machine vision was used. The exact algorithm using machine vision for longer-range detection used the intermediate algorithm of laser detection. The terrain up to 80 m was mapped by the vision system. The laser system would map the terrain up to 22 m in front of the vehicle, in such a way that the terrain mapped was considered drivable. This drivable terrain is then used as a projection to map all the pixels considered drivable or not drivable in the 80 m image.
   The algorithm uses a mixture of Gaussian models to map the terrain in front of it. It denotes each mixture with a number of images 'n', a mean value and a covariance value. Each time a new image is detected, its Gaussian model is extracted with new values of 'n', mean and covariance. This new Gaussian model helps in the learning algorithm in two different methods:
   a. Slow adaptation – The previously found Gaussian properties are adjusted to the newly found image's pixels. This enables the vehicle to adapt to slowly changing light conditions.
   b. Fast adaptation – This method discards previous Gaussians and introduces new ones. It allows the vehicle to adapt quickly to a surface with a new color.

   For detecting the drivable surface of the road, the newly obtained image pixels are compared with the model of learned Gaussians already present in memory. If the pixels closely match any of the Gaussians, the surface is considered to be drivable, else not drivable.

2. Use of LIDAR: Laser Detection and Ranging is used in Stanley to find obstacles in the short and medium range. It does so by generating a cloud of 2-D points in front of the vehicle. Each location of points is assigned one of three values, occupied, free or unknown. The location is classified as occupied if the vertical distance is beyond a certain threshold, and if no such threshold is crossed, the location is classified as free. What makes the algorithm more efficient is the fact that only two-point values need to be stored for each grid cell, each taken after a specific time interval. The difference between these values is what gives the calculation for obstacles in front.
   As to whether LIDAR could have been fully replaced by a sophisticated machine vision system with multiple cameras, it does not seem possible with the current algorithm of Stanley. The computer vision algorithm serves to extend the range of the LIDAR algorithm. The main reasons

why computer vision could not be used primarily and solely in this application are that with changing visibility, dust on camera lens, lighting and terrain conditions, computer vision is not so reliable. It is only used to augment the effects of LIDAR. One example of how computer vision can sometimes be the cause of inaccuracies is if we consider a road in front of the vehicle, along with a grass trac to the side. Due to the color of the grass surface being different from a surface that is considered drivable, the grass will be classified as undrivable, even if it is not. For this reason, it is not feasible to consider replacing the LIDAR system by a machine vision system, without significantly changing the algorithm used by Stanley.

Question 2

Using the HoG features in this code of pedestrian detection was a given, considering it is still one the most reliable methods for doing so.

For the purpose of this program, I used OpenCV's in-built pre-trained HoG model. As it is, I have used the 'setSVMDetector' API to set the detector to detect people by default. A loop is then run, in which the 'detectMultiScale' API is used to return a vector of rectangles, each rectangle containing a detected person's features. A bounding box is drawn around this rectangle, and the video is displayed and saved.

```
sarthak@sarthak-nano:~/Desktop/EMVIA_SU'20/Project$ make
g++ -O0 -g   -c detect_person.cpp
g++  -O0 -g   -o detect_person detect_person.o `pkg-config --libs opencv` -L/usr
/lib -lopencv_core -lopencv_flann -lopencv_video
sarthak@sarthak-nano:~/Desktop/EMVIA_SU'20/Project$ xdg-open output.avi
sarthak@sarthak-nano:~/Desktop/EMVIA_SU'20/Project$ make
make: Nothing to be done for 'all'.
sarthak@sarthak-nano:~/Desktop/EMVIA_SU'20/Project$ ./detect_person Videos/S3V8.
mp4
Gtk-Message: 22:31:07.218: Failed to load module "canberra-gtk-module"
OpenCV: FFMPEG: tag 0x5634504d/'MP4V' is not supported with codec id 13 and form
at 'avi / AVI (Audio Video Interleaved)'
OpenCV: FFMPEG: fallback to use tag 0x34504d46/'FMP4'
Number of frames: 1842
Duration: 277 seconds
Average FPS: 6
```

Build and run on video 1.

```
sarthak@sarthak-nano:~/Desktop/EMVIA_SU'20/Project$ ./detect_person Videos/S0V1.
mp4
Gtk-Message: 22:56:25.641: Failed to load module "canberra-gtk-module"
OpenCV: FFMPEG: tag 0x5634504d/'MP4V' is not supported with codec id 13 and form
at 'avi / AVI (Audio Video Interleaved)'
OpenCV: FFMPEG: fallback to use tag 0x34504d46/'FMP4'
Number of frames: 1844
Duration: 278 seconds
Average FPS: 6
```

Run on video 2.

The logic of the program is simple and straightforward enough, it is the fine-tuning of the parameters which is crucial. The main things the parameters of the 'DetectMultiScale' can affect are the number of false detections, entire misses in detection, and speed of the detection process. The way this API works is by sliding a step window over the range of the entire image, and the size of this sliding window is one aspect controlling the speed of the program. The larger the window, the less it would have to slide over the image. However, in doing so, it might also miss objects entirely if they are smaller than the window size. I have settled on a window size of 4x4 after much testing.

The parameter of 'scale' controls the number of layers n the image pyramid. A larger scale size leads to evaluation of less layers in the image pyramid, making the algorithm faster. I have settled on an optimum value of 1.05 for scale. These are the main parameters controlling the speed and accuracy of detection in this program, and I have believed I have achieved a competent rate of detection without the use of ML algorithms.

Output Video analysis

I have recorded two videos and processed them using my algorithm. Both videos have a consistent FPS of around 6, when the input video has an FPS of around 30. In the first, the car is stationary almost all of the time at a zebra crossing, and plenty of pedestrians can be seen crossing the road. Each one of them is detected, and the program is able to follow them throughout the entire frame of the program. However, this detection only works on pedestrians directly in front of the frame, not on pedestrians in the peripheral vision of the video. I would assume that this detection would work better with a camera of better resolution. In the second video, the car is moving through an airport road, sometimes detecting pedestrians around the periphery, but mostly falsely detecting reflections off of cars.

To sum up, the program works sufficiently well on pedestrians directly in front of the car, but not on pedestrians too far away. This can be amended by using a video of better quality, and employing Machine Learning techniques using the linear SVM model which the HoG classifier is already pre-trained for.

Question 3

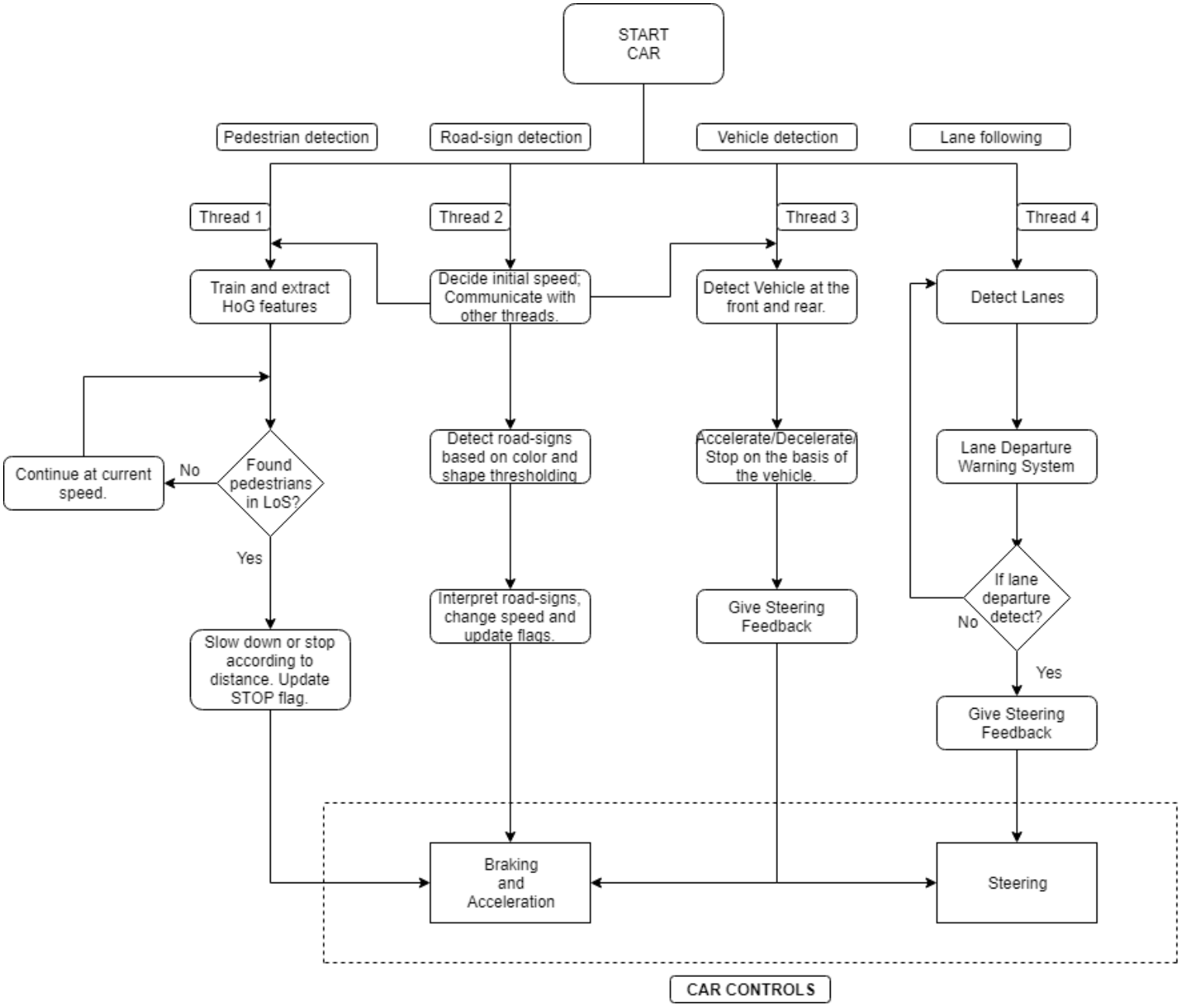
Self-driving car using pedestrian detection, lane following, road-sign recognition and vehicle detection –

1. Relevance of selection: The topic I have selected is highly relevant to the field of computer vision. It uses almost all of the techniques we have used previously during the course, and optimizing them so as to make use of them all, while at the same time integrating them to make sure they are each able to perform their desired function.
   The final program will involve using image processing transforms for preprocessing in order to clean up each subsequent frame to remove unwanted objects. Noise will be removed using either Gaussian filters or the median filtering techniques. Subsequent processing will involve detection of each particular object on different threads, employing multi-processing capabilities of the Jetson. The final objective is to reach a stage where the program can emulate to some degree a self-driving car, by slowing down when it detects pedestrians, follow a lane by tracking the curves of the lines marking roads, detect signs and react appropriately, and detect vehicles in front as well, to slow down if the vehicle in front is too close, or accelerate within speed limits given no vehicle is in front.
2. The main challenges during the development of this program will be:
   a. Maintaining a feasible frame rate: Each processing stage is highly computationally intensive, and when run on a single core, gives an extremely low rate of Frames/second. Using multithreading and possible code optimization to speed up this processing will be one of the main challenges.
   b. Accurate detection of objects/targets: In the early testing stages, often times the program would detect a reflection or a side-object as the target rather than the intended target itself. Modifying and correcting the program will be a large part of the testing, in order to obtain the correct parameters, or even sequence of program which successfully detects its targets.
   c. Integrating all different sections to obtain one complete program. Ensuring the processing of one part does not interfere with the other parts will be key to ensuring a successful program. Allocating CPU resources as well to ensure the output is such that the car has enough time to react is also important.
   d. Possible integration of machine learning algorithms to improve efficiency: Using machine learning techniques is still under consideration, but if the image processing and machine vision techniques are unable to produce the desired level of accuracy, some machine learning elements may have to be included in order to obtain better efficiency.

3. Rationale behind selection: A large number of the corporate giants is working on the problem of self-driving cars, without having a marketable prototype at this stage. However, each one of the systems of these companies will only be available to the cream of the market, not being available for the common man during the early stages. The program we are working on will be a low-cost software, having the ability to be integrated with any car having a sufficient set of requirements.
   Speaking on a smaller scale, the rationale behind selecting this topic is the chance to integrate all the skills acquired previously during the course, along with the opportunity to look into advanced machine vision algorithms and possibly machine learning algorithms as well.

Question 4

The program shall have four main components to with scope for addition of more: the pedestrian detection unit, the road-sign recognition unit, the lane following unit and the vehicle detection component.



The minimum set of requirements for the project to be considered complete is:

1. Minimum requirements –
    a. Pedestrian detection: The minimum requirement is the detection of all pedestrians in a particular shot of video, without any being missed. The HoG feature extractor currently detects most people in a frame, but at times tends to skip out, based on the lighting and camera conditions. Sometimes it catches false features as well, like the reflection off of a car, or the glare of the sky. Most of this can be attributed to picture quality. However, a final end point of this will be to obtain a program which can successfully recognize all characters in the frame without skipping any people. Work around for the false detections can be found, by using techniques like Non-Maxima Suppression.
    b. Road-sign detection: The minimum requirement would be to detect a basic set of road-signs, and recognize the more important ones, like “Stop”. Using Haar feature extractors, the camera should eventually be able to recognize the “Stop” sign, even when travelling at a relatively high speed from afar in order to send the command for bringing the vehicle to a halt.
    c. Lane following: The minimum requirement would be to detect lanes, even if rudimentarily, along a clearly demarcated road. This program would use Canny and Hough-line transforms, so detecting only the lines would be the tricky part of this unit. Reaching the point where lines can be recognized, even if not all of the time, but sufficiently enough to allow the car to stay within boundaries, would be required.
    d. Vehicle detection: This part of the program would use Haar feature extractors to detect vehicles in front, and as a minimum requirement, would send a STOP command to the control unit if the vehicle in front gets too close, else continue moving if no vehicle is detected in front.

2. Target requirements –
    a. Pedestrian detection: The target is to bring the FPS of detecting pedestrians up to an optimum level. Currently the video plays at an FPS of around 30, and processing gives an FPS of less than 7. The target would be to bring this processing up to at least 20, using multithreaded concepts, and possibly program optimization. Another important section in this part is better recognition of pedestrians, up to the order that false detections are minimized, and only pedestrians are detected.
    b. Road-sign detection: Recognition of more signs, along with speed limits would be desired, in order to keep the car well within safety limits. For this, recognition of numbers would also be required, which can be achieved by machine learning algorithms.
    c. Lane following: Better detection of lanes, along with determining the center point of lines and steering the car to following the central lined would be the desired level of accuracy to be achieved. Testing the program in harsher conditions, like on a dirt track with less clear demarcations would be a good stage to achieve.
    d. Vehicle detection: Detecting vehicles in front, and implementing an algorithm to follow them along a straight would be similar to turning on autopilot, when it is known that no turn is to be taken. This would be an interesting feature to implement, and would make the final execution of the program that much easier.

3. Optimum requirements –
    a. Pedestrian detection: Eventually this unit would be used to control the braking and acceleration unit of the car. The optimum level to reach would be to detect pedestrians even in the peripheral field of view, and calculating their speed and direction of movement to give pseudo-predictions as to whether their movements would bring them on a collision course with the heading of the car.
    b. Road-sign detection: Optimizing performance of this unit so not to put too much of a high load on the processing capabilities of the computer would be important. This unit would eventually control the braking and acceleration of the car as well. It is expected that this unit along with the pedestrian detection unit would be most computationally intensive, and as such their optimization and CPU analysis will be important.
    c. Lane following: Optimization of program to ensure storage capabilities are not exceeded would be important here. This program would control the steering unit of the car.
    d. Vehicle detection: This program would control the steering and braking-acceleration units of the car.

References:

1. Hog detectMultiScale parameters explained: https://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/
2. Hog descriptors: https://docs.opencv.org/2.4/modules/gpu/doc/object_detection.html
3. Pedestrian detection in OpenCV: https://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/
4. Histograms of Oriented Gradients for Human Detection(Triggs-Dalal): https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf