CLEMSON UNIVERSITY INTERNATIONAL CENTER FOR AUTOMOTIVE RESEARCH

Name: Vishal Jadhav

Course: 8240 AUTONOMOUS DRIVING TECHNOLOGIES

Project: Autonomous Navigation on Road

Spring 2022

Instructor: Dr. Yunyi Jia

Section 1: Problem Statement

The project's problem statement includes completing 4 different tasks and integrating it. The four tasks are as follows:

1. Autonomous Lane Keeping:

Autonomously track the lane clockwise using a camera and a laptop as fast as possible.

2. Recognize road signs:

Autonomously recognize a stop sign and a school zone using a second camera and a laptop.

3. Communications:

Send the road sign information from the second laptop to first laptop through Wi-Fi communications.

4. Vehicle Controls:

Stop the vehicle at the stop sign for 2 seconds and then run at high speed; Reduce the speed by half at the school zone sign.

Section 2:

Autonomous Lane Keeping

Technical Approach

1. Camera Calibration

Camera calibration is estimating the camera parameters using images of a special calibration pattern. Camera calibration is the process of estimating intrinsic and extrinsic parameters. Intrinsic parameters deal with the camera's internal characteristics, such as its focal length, skew, distortion, and image center. Extrinsic parameters describe its position and orientation in the world. Knowing intrinsic parameters is an essential first step for 3D computer vision, as it allows you to estimate the scene's structure in Euclidean space and removes lens distortion, which degrades accuracy.

Camera calibration is done using MATLAB's Camera Calibrator app. We need to prepare images of checkerboard patterns from different orientations for camera calibration. The checkerboard pattern you use must not be square. One side must contain an even number of squares and the other side must have an odd number of squares. Therefore, the pattern contains two black corners along one side and two white corners on the opposite side. After capturing the images, add the images to the app and select the standard camera model. Then calibration is done using the option of calibrating. MATLAB will automatically remove bad images, and calibration is done using the remaining images. Examine the reprojection errors. The Camera Calibrator app calculates reprojection errors by projecting the checkerboard points from world coordinates, defined by the checkerboard, into image coordinates. After examining extrinsic parameter visualization, view the undistorted image to check if the distorted lines become straight.

Image preparation for camera calibration

• Attach the checkerboard printout to a flat surface.

• Measure the size of the checkerboard square accurately.

• Keep the pattern in focus, but do not use autofocus.

• Do not change zoom settings between images.

• For best results, use about 10 to 20 images of the calibration pattern.

• Place the checkerboard at an angle less than 45 degrees relative to the camera plane.

• Do not modify the images.

• Capture the images at different orientations relative to the camera.

• Capture enough different images of the pattern so that you have covered as much of the image frame as possible.

• Checkerboard should fill at least 20% of the captured image.

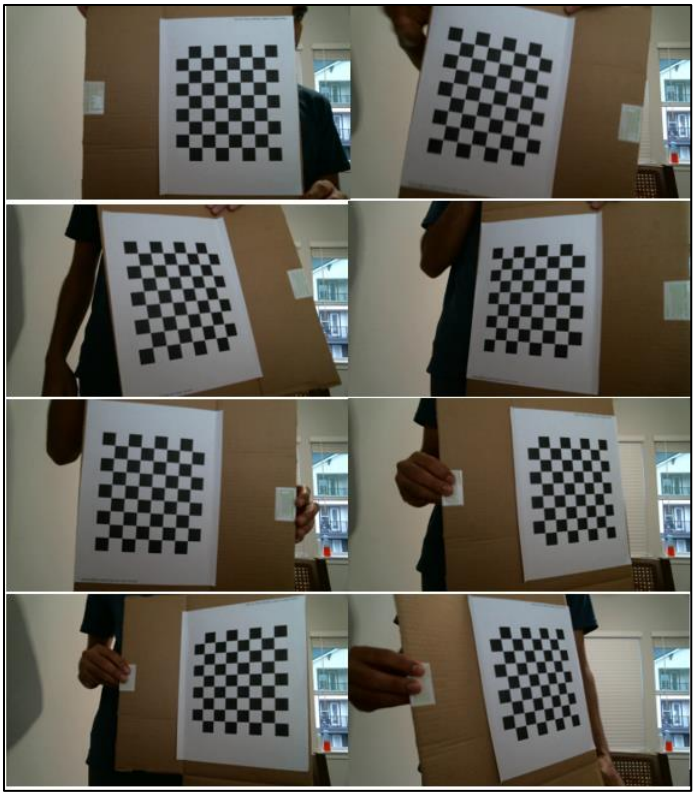

Fig. 1 Various orientation of checkerboard pattern for calibration (6 Images)

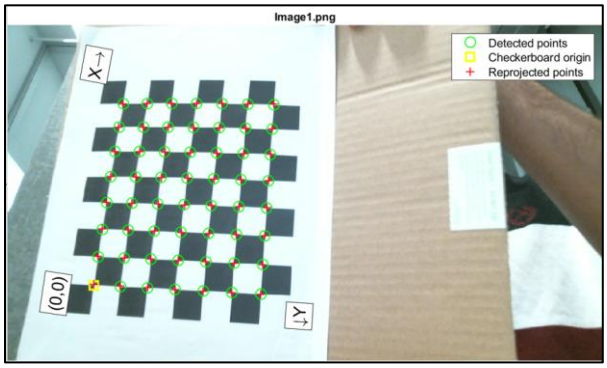

Fig. 2a Distorted Image before calibration

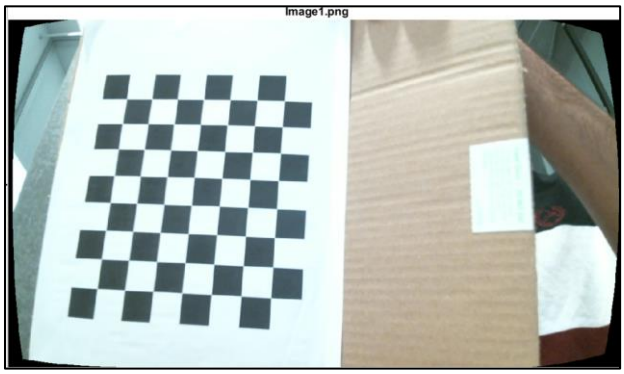

Fig.2b Undistorted Image after calibration

The Parameters Found After Calibration:

cameraParams11.Intrinsics

Property	Value
FocalLength	[941.4417,942.5781]
PrincipalPoint	[576.1953,367.1024]
ImageSize	[720,1280]
RadialDistortion	[-0.2370,0.2545]
TangentialDistortion	[7.4320e-04,0.0053]
Skew	0
IntrinsicMatrix	[941.4417,0,0;0,942.5781,0;576.1953,367.1024,1]

Fig. 3 Camera Intrinsic Parameters

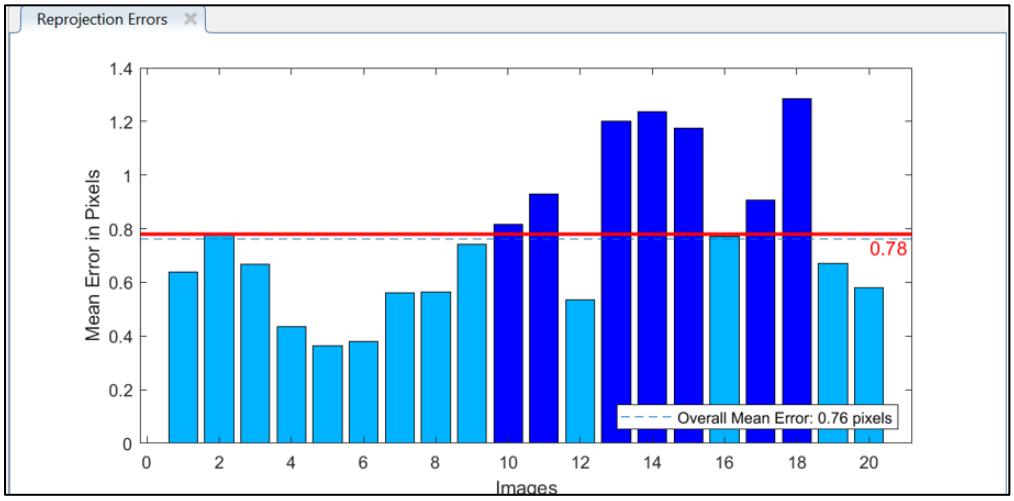

Fig. 4 Mean Reprojection Error

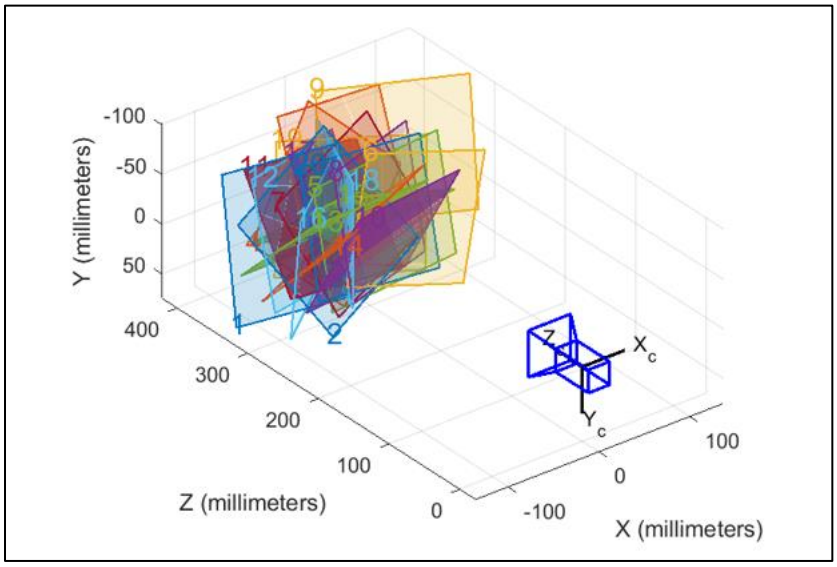

Fig. 5 Extrinsic Parameters Visualization

2. Detection the Track

There are two ways to detect the lane lines. One uses hough transform, and the other is blob detection (HSV) transformation.

Edge Detection

Implementation of track detection using edge detection can be seen as below.

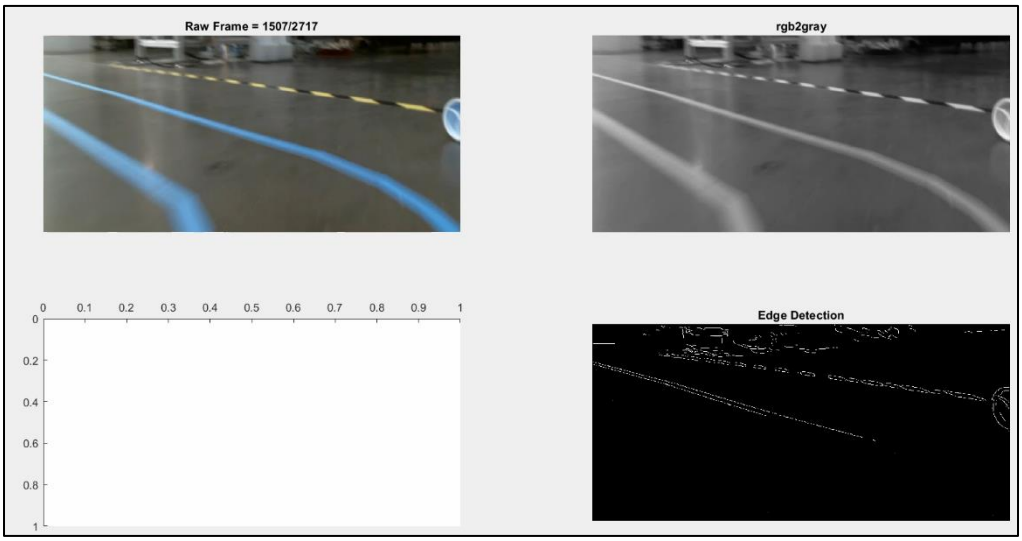

Fig. 6 Edge Detection using Canny Edge Detector

The Canny Detector is a multi-stage algorithm optimized for fast real-time edge detection. The fundamental goal of the algorithm is to detect sharp changes in luminosity (large gradients), such as a shift from white to black, and defines them as edges, given a set of thresholds. It has 4 stages viz Noise reduction, Intensity gradient, non-maximum suppression, Hysteresis thresholding.

MATLAB has a tool called computer vision and it has function for edge detection called edge. only Canny and approxcanny can apply thresholds to get rid of edges whose intensity does not match our track. Therefore, Canny will be used to get more accurate track detection. Here is where the first problem with edge detection lies. Even if we tune the upper and lower threshold to detect the track perfectly, many other edges happen to fall into that range and show up. This can be seen in the bottom right corner of fig 6 as artifacts or non-track detected objects. The other problem with edge detection is that edge can only take a grayscale image as an input. This means we must convert our raw colorful frame to black and white. This conversion can be seen in the top right corner of fig 6. A lot of information which could be used to detect the track more robustly is lost in this step.

Color Masking

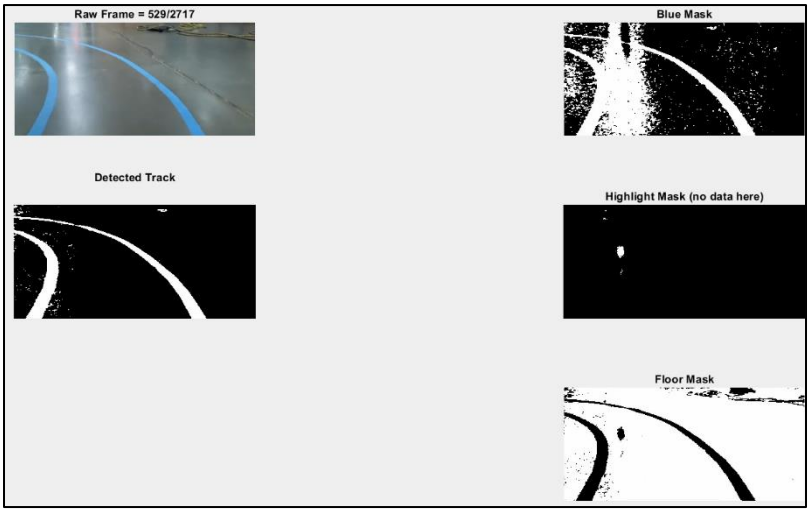

Fig. 7 Implementation of color masking

Implementation of color masking is shown in above figure. Instead of using RCB colorspace we can use HSV color space. Matlab convert RGB to HSV using rgb2hsv function. It produces an image whose first matrix is the hue values, second matrix is the saturation values and third matrix is the value values. All of these are normalized from 0 to 1. HSV color space can be seen in below fig.

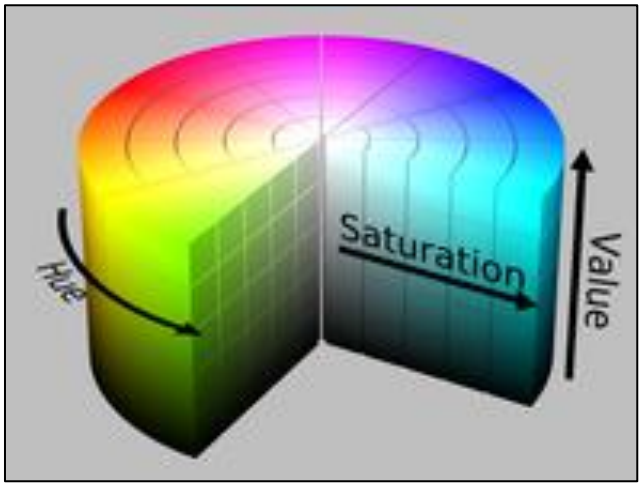

Fig. 8 HSV color space.

In HSV colorspace, find the blue track's hue then +/- some value to that tolerance. This mask can be seen at the top right corner of Fig. Here white is where it has detected the blue hue and black is not. Code implementation can be seen below.

```
%mask hue channel (detect blue of track)
mask1 = convert(:,:,1) < 1*(210/360) & convert(:,:,1) > 1*(187/360)& convert(:,:,2) > 0.2;
```

Fig. 9 Hue mask code implementation

As you can see that mask is noisy. To filter this noise we add another mask to ignore the highlights. This can be seen as the middle tight subplot of fig 7. The code implementation can be seen below.

```
mask2 = ~(convert(:,:,2) <= 0.1 & convert(:,:,3) >= 0.85);
```

Fig. 10 Highlight mask code implementation

~ here ignore what this mask detects.

Above two mask create decent results. So we try to detect gray of floor and ignore it to further clean up the final output. This mask can be seen in bottom right corner of fig 7. The code implementation can be seen below.

```
%remove gray of garage floor
mask3 = ~(convert(:,:,2) <= 0.2 & convert(:,:,3) < 0.90 & convert(:,:,3) >= 0.05);
```

Fig. 11 Remove gray of floor

Now combine all three masks simply by element multiplying altogether. This can be seen in the middle-left of fig 7. The code implementation can be seen below.

```
mask4 = mask1 .* mask2 .* mask3;
```

Fig. 12

Following the Track

Now that our car knows where the lanes are, it needs to know how to follow them. We solved this by averaging method. Implementation of averaging method can be seen in below fig.

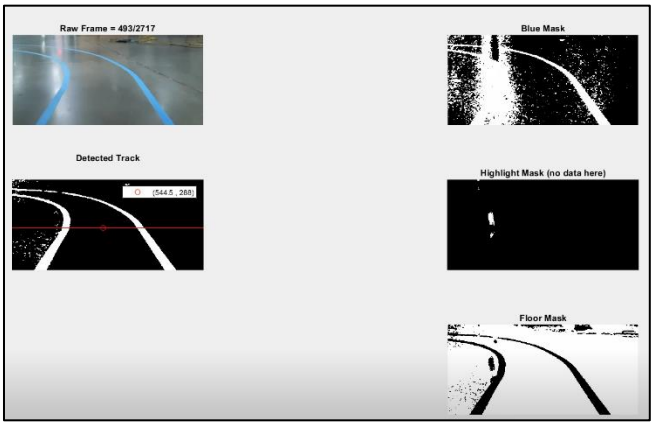

Fig. 13 Averaging method implementation

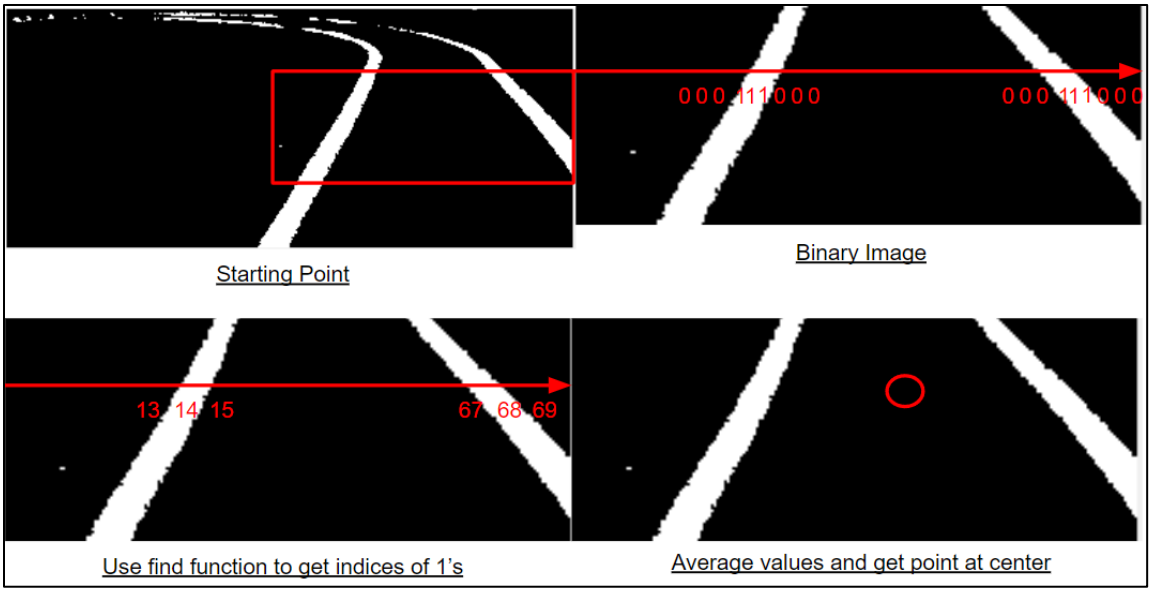

Fig. 14 How the averaging method works

First, we take horizontal line in our color masking output binary image. The horizontal line is our look ahead distance. Along the line it's either o or 1. On the track line detects 0 and other location as 0. We can now use MATLAB's find function to detect the indices of all the 1's. The indices in this case are our x-location pixel values. Our y-location pixel values are all fixed at whatever height we setup our horizontal line. So now we know where on our frame our track is. If we take the average of all of these pixel values, we get the center of the track along that horizontal line. The code which performs these steps can be seen in below figure.

```
dims = [1080, 1920];

threshold = 0.3; %how far up from bottom of image to average lanes
y = round(dims(1)*(1-threshold));
```

```
inds = find(mask4(y,:));
x = mean(inds);

plot(x,y,'ro');
```

Fig. 15 Code implementation of averaging method

This method will give around 200 points that are track or maynbe 20/20 points that could be artifact. When we take the average of all these, the correct points largely outweigh the artifacts and the center of track remains very close to its theoretical position.

One drawback of this method is that when there is only one lane line at horizontal line height, the car starts to follow only the lane it can see. This mostly happens in turns. This is partially solved by center point of a lane being close to the middle of the frame due to the lookahead distance. We

also mounted the camera as high up as we could to increase its field of view. We could not go any higher because of the build space of the 3d printer we used. In real tests, the car was always ¾ between both lanes even when this occurred.

3. Controller

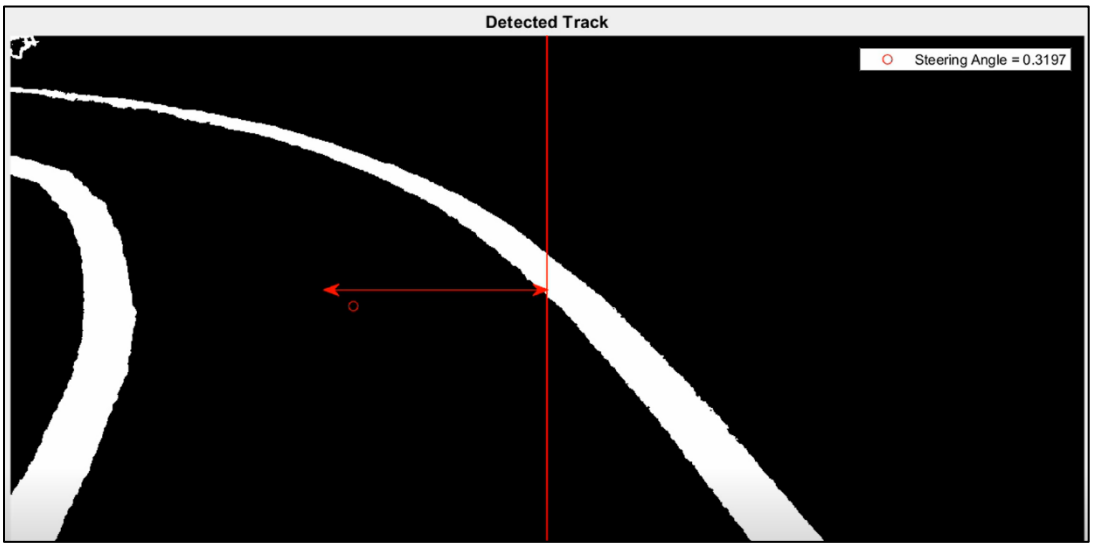

Fig. 16 Steering angle based on averaging metod

We used simple P controller to follow detected point. The controller can now easily be designed. If the center of the track is in the center of frame, don't steer. Else, steer left or right linearly based on how far the center of the track is from the center of the frame. This distance must be normalized between 0 and 1 because that is the input the servo expects. In above fig, at upper right corner is steering angle. The code of implementation can be seen below.

```
midpoint = dims(2)/2;

delta = (((x-midpoint)/midpoint)+1)/2;%range [0,1]

writePosition(steeringPin, delta);
```

Fig. 17 Code implementation for steering angle

Now, vehicle can detect track and follow the track. Next step in vehicle should recognize road sign and act accordingly of road signs.

Section 3. Sign Recognition and Vehicle Controls

With using second camera on vehicle, the task is to recognize the stop sign and school zone sign mark them using bounding boxes. The video was stopping near the stop sign for 2 seconds and in school zone it is slowing down to its half speed. So, sign recognition important only where it was stopping and slowing down.

3.1. Technical Approach

There are 2 approaches with which we can do sign detection

1. Traditional Machine learning –

Cascade Object Detector

• The cascade classifier consists of stages, where each stage is an ensemble of weak learners.

• Each stage of the classifier labels the region defined by the current location of the sliding window as either positive or negative. Positive indicates that an object was found, and negative indicates no objects were found.

• If the label is negative, the classification of this region is complete, and the detector slides the window to the next location.

• If the label is positive, the classifier passes the region to the next stage.

• The detector reports an object found at the current window location when the final stage classifies the region as positive.

• Cascade Object Detector supports three types of features: Haar, local binary patterns (LBP), and histograms of oriented gradients (HOG).

• Haar and LBP features are often used to detect faces because they work well for representing fine-scale textures.

• The HOG features are often used to detect more general objects such as people, cars and traffic signs. They are useful for capturing the overall shape of an object.

Process and properties

• The Cascade object detector can detect object categories whose aspect ratio does not vary significantly. Objects whose aspect ratio remains fixed include faces, traffic signs, and cars viewed from one side.

• The detector detects objects in images by sliding a window over the image. The detector then uses a cascade classifier to decide whether the window contains the object of interest. The size of the window varies to detect objects at different scales, but its aspect ratio remains fixed.

• The detector is very sensitive to out-of-plane rotation because the aspect ratio changes for most 3-D objects. Thus, you need to train a detector for each orientation of the object.

As there are some disadvantages which are highlighted above as well as it requires negative images apart from positive images and the number of images required is high as compared to other approach. So, we preferred second approach which will be discussed in the next section.

2. Deep learning –

Convolutional Neural Network (CNN) Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars. The approach used is transfer learning which has several advantages over traditional approach as follows.

• Fine-tuning a pretrained network with transfer learning is typically much faster and easier than training from scratch. It requires the least amount of data and computational resources.

• Transfer learning uses knowledge from one type of problem to solve similar problems. You start with a pretrained network and use it to learn a new task.

• One advantage of transfer learning is that the pretrained network has already learned a rich set of features. These features can be applied to a wide range of other similar tasks. For example, you can take a network trained on millions of images and retrain it for new object classification using only hundreds of images.

Region-based Convolutional Neural Network (R-CNN)

• The R-CNN first generates region proposals using an algorithm such as Edge Boxes. The proposal regions are cropped out of the image and resized.

• Then, the CNN classifies the cropped and resized regions.

• Finally, the region proposal bounding boxes are refined by a support vector machine (SVM) that is trained using CNN features.

• R-CNN is faster in classification than CNN because of using cropped small-region images

R-CNN Transfer Learning using CIFAR10Net

• CIFAR10Net is a convolutional neural network trained using the CIFAR 10 dataset.

• The model has two convolutional layers and two fully connected layers. Each convolutional layer uses the leaky ReLU activation function and is followed by a pool layer of size 2 and stride 2.

• The fully connected layers use the sigmoid activation function to coerce outputs to [0, 1].

• They also have droupout with probability 0.5.

• The output layer has 10 neurons, each representing the probability the image belongs to that class.

Implementation steps

Implementation of stop sign detection

1. Create training data by capturing the images of stop sign and school sign. We did it by recoding video around track and taking images from the frames.

2. Then we labelled the image via image labeler app which is MATLAB's own toolbox. In that we create 2 labels and drew bounding box around each sign in each image.

3. Then we exported the labels in MATLAB workspace.

4. Then we trained RCNN model on Cifar10 dataset.

5. Use this trained network for transfer learning. By using training and testing dataset we labeled, I train the network again for two classes stop sign and school sign.

5. After training we tested the network on sample images to check the network accuracy whether its working properly or not.

6. Then we deploy that model on vehicle. On vehicle we tune the parameter like probability and area of sign in controller.

Training parameters:

1. Number of images used for training were 408. (Stop sign = 188 and school sign = 220)

2. We used SGDM optimizer.

3. Batch size was 128 images.

4. Learning rate schedule was Piecewise.

5. Epochs used were 100.

6. We used 0.001 learning rate.

7. And we set Verbose to true to show any warning and log labels.

```
trainingData = objectDetectorTrainingData(gTruth);

options = trainingOptions('sgdm', ...
'MiniBatchSize', 128, ...
'InitialLearnRate', 1e-3, ...
'LearnRateSchedule', 'piecewise', ...
'LearnRateDropFactor', 0.1, ...
'LearnRateDropPeriod', 100, ...
'MaxEpochs', 100, ...
'Verbose', true);

rcnn = trainRCNNObjectDetector(trainingData, cifar10Net, options, ...
'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1])
```

Fig 18. Hyperparameters of model

Total time required to train the network was 90 minutes because of it was trained on the CPU.

3.2. Challenges faced while doing this task

• Initially, we tried training the network with 111 stops sign images and 103 school sign images. The results were that the neural network was not able to detect the images when tested.

Fig. 19 Unable to detect stop sign

• This problem may be faced due to the quality of the images and less images used to train the network.

3.3. Solution implemented

- This problem was solved by training the network using a large number of images and also a wide variety of images with different angles.
- The quality of the images was also varied, and the network was trained for blurry as well as clear images .
- We faced issue due to overfitting too. In that case we lower the number of epochs.

3.4. Experimental Results

Fig. 20 Detect stop sign

Detection of a stop sign with a bounding box drawn in above figure.

Fig. 21 Detection of school sign with bounding box drawn

We can see the results in the images as stop sign and school sign is detected with confidence level and we can see the labels in the image with bounding box as well.

The results showed high confidence in our detection and the bounding box allowed us to see what was being detected for debugging during full stack testing.

4. Communication

Communication between laptop is done using UDP protocol.

```
ipA = '198.21.134.191';    portA = 4090;
ipB = '198.21.190.35';    portB = 4091;
udpB = udp(ipA,portA,'LocalPort',portB);
fopen(udpB);
```

Fig. 22 UDP communication code from second laptop.

Section 4 Discussions on Achievements and Challenges

Our vehicle able to complete three laps around track successfully. Below figure is demonstration of vehicle on track.

Fig. 23 Vehicle stop at STOP sign

I discussed lot of challenges in the respective section of lane detection, stop sign detection and controller.

Challenges:

1. Designing PWM for the motor. One, at the lowest torque setting of the motor, the car was already accelerating like a bullet.

2. Tires of vehicle designed for mud. It was slipping on the floor.

3. Parallelize code to not run just on one core. We were getting only 1.5 fps. Would allow much faster speed.

4. Faced issues while setting up UDP communication between two laptops.

5. HSV parameters can’t be kept constant for all lighting conditions.

Clemson University : Survey Certificate of Completion
202201 Spring 2022 Evaluations

Course: AUE-8240-001 : Autonomous Driving Technol : 202201-AUE-8240-001-17855-Jia
Instructor: Yunyi Jia

Submitted: 4/29/2022 12:20 AM
Student: Vishal Jadhav

Thank you for completing a survey!