Intelligent Transportation using Deep Learning

Vijay Bhanudas Gujar

Abstract: ***The goal of this paper is to advance intelligent transportation program through the creation of a data collection system, a Convolutional Neural Network (CNN) model for intelligent transportation, and a simulator to test the trained CNN model. The data collection system collects data from a vehicle-steering wheel angle, speed, and images of the road from three separate angles at the time of the data collection. A CNN model is then trained with the collected data. The trained CNN model is then tested on a simulator to evaluate its effectiveness.***

Keywords: Convolutional Neural Network (CNN), Data Collection System, Deep Learning, Neural Network, Simulation.

I. INTRODUCTION

The field of intelligent transportation is currently one of the most prominent and popular fields in the industry, it is a field with lots of room to grow and develop. This paper creates a wide variety of possible future directions. The first possible direction being the completion of the real world 3D simulator, at task that proved to be out of the scope of this project-however it is a task that would be well suited as the focus of a future project building on the progress completed here. Another obvious and compelling direction being the implementation of the trained CNN model not only on a simulator but also a real active vehicle. Furthermore, a small car model could also be used to collect and test data from, barring access to a fully sized active vehicle. The goal of this project was to advance intelligent transportation program through the creation of a data collection system, a Convolutional Neural Network (CNN) model for intelligent transportation, and a simulator to test the trained CNN model. By developing these tools, it was our aim to further enhance, advance, and aid intelligent transportation program.

A. Current State of the field

Intelligent transportation has become a very prominent field in the automotive industry. This has led to a huge influx in funds allocated to researching and developing new and more efficient technologies for intelligent transportation, Convolutional Neural Network models, and intelligent transportation simulators similar to this paper.

Data from three front facing cameras and a vehicle's steering wheel angle; both project then proceed to feed this information to a neural network that utilizes pattern recognition to train itself. However, the two projects differ in terms of simulation. While NVIDIA utilizes prerecorded video to approximate how its network model would operate, this paper evaluates its progress through the use of a 3D real time simulation. In addition, this paper is also capable of 3 training its neural network through the use of data collected in-simulation as well as real world data.

B. Proposed Design and Contributions

This paper was composed of three major core components: (1) the creation of data collection system, (2) a Convolutional Neural Network (CNN) model, and (3) a simulator. (1) The data collection system proposed would be used to collect data from an active vehicle in real time, it would collect the steering wheel angle, speed, and images of the road from three separate angles. (2) The CNN model proposed would then be trained with the collected data from the vehicle. (3) The simulator would test/evaluate the effectiveness of the data collection system trained CNN model in a safe and controlled environment.

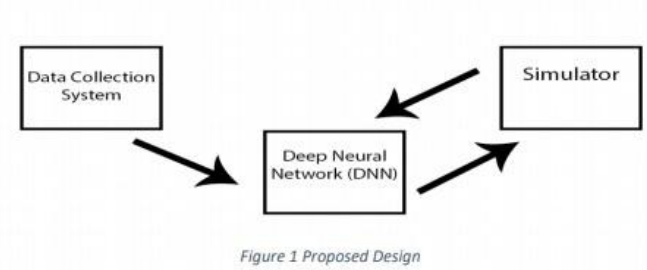


Figure 1 Proposed Design

The result of this paper was proposed to be a modular collective system for intelligent transportation that could be implemented and tested in a variety of different vehicles to collect data, create models based of the data, and then test the data in the controlled environment of a simulator.

C. Main Goal

The goal of this paper was to advance intelligent transportation program through the creation of a data collection system, a Convolutional Neural Network (CNN) model for intelligent transportation, and a simulator to test the trained CNN model. By developing these tools, it was our aim to further enhance, advance, and aid intelligent transportation program. This paper was composed of three core components: the data collection system, a Convolutional Neural Network (CNN) model, and a simulator.

The data collection system was used to collect data from an active vehicle in real time; it collected the steering wheel angle, speed, and images of the road from three separate angles. The CNN model was then trained with the collected data from the vehicle or with available datasets online. Then, having collected the data and trained a CNN model with it,

the trained model was then tested in a simulator to evaluate its effectiveness in a safe and controlled environment.

Revised Manuscript Received on January 05, 2020

Mr. Vijay Bhanudasd Gujar, CSE, Dnyanshree Institute of Engineering and Technology, Satara, Maharashtra, India.

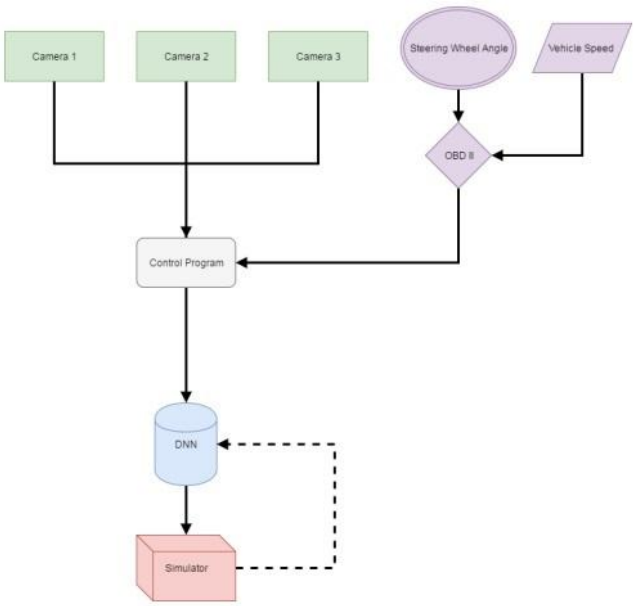


Figure 2 Main Goal

The main objective of this project was to create a modular collective system for intelligent transportation that could be implemented and tested in a variety of different vehicles to collect data, create models based of the data, and then test the data in the controlled environment of a simulator. This objective was broken down into three sub objectives.1 Data Collection System Create a data collection system modular in nature that can be placed on a test vehicle, used to collect data, then removed and taken back to the lab for either further 7 in-house testing, modifications, repairs, or safekeeping. Attaching and removing this data collection system should be quick and easy to accomplish. 2. Convolutional Neural Network Model Develop a Convolutional Neural Network model that can be trained to operate a vehicle through real data gathered by the data collection system. The Convolutional Neural Network should also be able to be trained by simulated data created in a simulation. The CNN model should be capable of navigating in clearly marked roads with good lighting conditions. 3 Simulator Develop a simulator to test a developed CNN model. The simulator should be capable of simulation a variety of driving conditions, obstacles, and road variations.

Data Collection System: Data Collection System Design When designing the data collection system one of the first design aspects that needed to be decided upon was the location of the cameras and the amount of cameras needed. While having cameras located throughout the perimeter of the vehicle (front, side, and back) would be beneficial for a production vehicle, the scope of this project would not permit it. Instead, the team decided to focus on the front of the vehicle for the data collection system. In this data collection system, three cameras would be placed near the front of the vehicle, providing ample coverage of the road in front of the vehicle. The cameras would be spaced out evenly on the vehicle, with one two cameras near the outer edge of the vehicle and one located in the center. This positioning of the cameras would provide the data collection system with a very wide view of the road when all three camera images were stitched together. final step in the data collection system design was determining a way of synchronizing the images collected by the cameras with the speed and steering wheel angle collected through the OBDII port.

In order to accomplish this, it was determined that the computer program for synchronizing the multiple cameras and the computer program for querying the vehicle for the speed and steering wheel angle would have to be merged together.

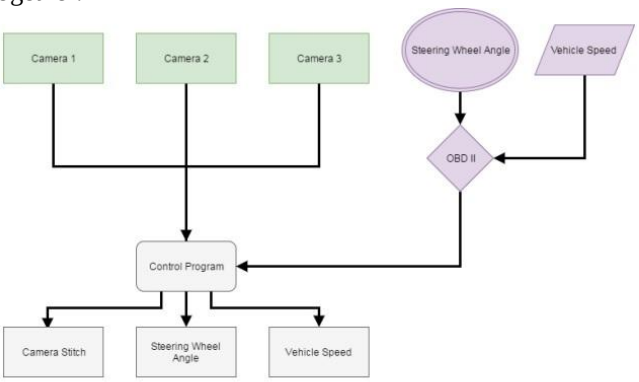


Figure 3 Data Collection System Overview

OBD II- On Board Diagnostics Background On-board diagnostics- commonly referred to as OBD- is a feature available in all modern vehicles which allow them to perform self-diagnosis and provide vehicle reports to the user and/or manufacturer. OBD provides the user with detailed vehicle information from a variety of topics including but not restricted to: car speed, steering wheel angle, fuel and air detection, ignition, emissions control, transmission control, and vehicle state (drive, reverse, park, neutral). Because of these capabilities, OBD technology is often used by car owners and car manufacturers to provide simple diagnostic codes which allow the user to quickly identify problems with the vehicle. OBDII are traditionally found and positioned in a location below the steering wheel and are usually hidden or covered by a removable compartment for ease of access. OBDII connectors are 16-Pin D-shaped connectors that transmit data over a CAN-bus protocol producing 4-digit hexadecimal PIDs (parameter IDs) for the user to read. While the method of transmission is standard, manufacturers are not required to standardize the description of each individual PID value. Resulting in a wide range of vehicle specific PID values that make it incredibly difficult to decipher the PID's description without direct information from the vehicle manufacturer.

II. CONVOLUTIONAL NEURAL NETWORK MODEL FOR LANE KEEPING

A. Introduction to End-to-End Learning for Lane Keeping

An end-to-end learning approach was implemented to obtain a steering wheel angle output based on an input frame. The Convolutional Neural Network model the team used was based on the network. The model was trained on a Udacity dataset and was evaluated by using the output to control the car in autonomous mode- this was done on the Udacity simulator at a constant speed.

The parameters of the neural network are optimized automatically using backpropagation based on the mean squared error between the

predicted angle and the labeled angle for the training image. Because the neural network is self-optimized, labeling the lanes on the image and extracting features manually is not needed, which makes pre-processing the images much easier. If training data for an abundant collection of road conditions are available, the prediction given by the neural network will be very accurate

In this portion of the project, the team set about to develop a Convolutional Neural Network model that could be trained to operate a vehicle through real data gathered by the data collection system and/or through pre-existing datasets. The goal of this portion of the project was to create a CNN model capable of navigating clearly marked roads or paths with good lighting conditions. The section of the project is broken up into two main sections. The first part details the creation of the CNN model as well as the training of the model. The second part consists of testing the CNN model created.

III. IMPLEMENTATION OF THE CONVOLUTIONAL NEURAL NETWORK

To implement this convolutional neural network, Keras, a high-level neural networks API, written in Python and running on top of Tensorflow, was used. The neural network structure was shown in the following image:

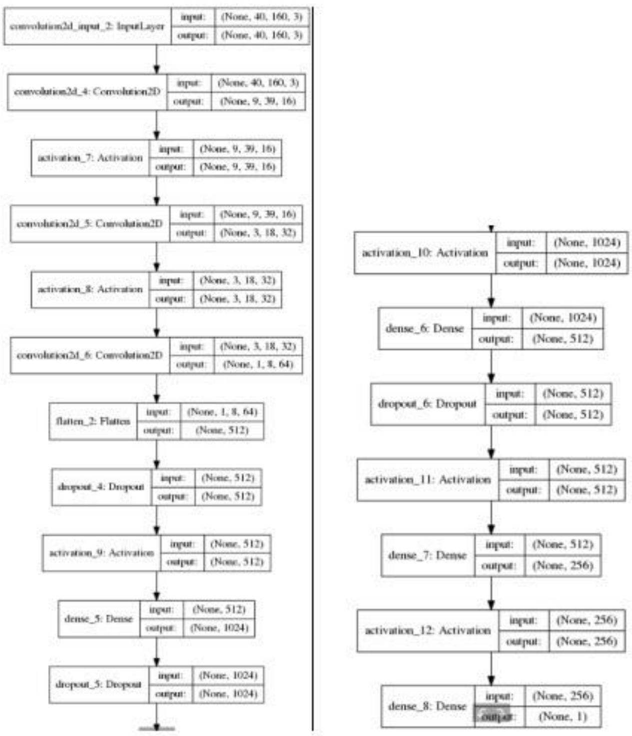


Figure 4 CNN Model

For this implementation, the input images were pre-processed before being fed into the neural network. First, in pre-processing, the upper half of the images were removed because the removed portion did not affect the result of lane keeping. Then the remaining images were shrunk by 0.5 to reduce memory usage and training time. Next, the images were converted from RGB to YUV. Finally, the images were adjusted to zero mean and unit variance to ensure 31 that convergence could be reached quickly. The code below shows the process of preprocessing images:

```
def processImg(img):
    yuv = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = img[80:160, :, :]
    img = ndimage.interpolation.zoom(img, [0.5, 0.5, 1])
    mean = np.mean(img)
    img = img - mean
    s = np.std(img)
    img = img / s
    return img
```

Figure 5 Image Pre-Processing

Since three images were captured by left, center and right cameras at each frame, the steering wheel angle corresponding to the left image was decreased by 0.25 and the angle corresponding to the right image was increased by 0.25. By doing this, the captured images were able to be fully utilized. The code used to implement the convolutional neural network is shown in the following image:

```
def cnn(inputshape):
    global conv1
    model = Sequential()
    # Batch size is none so the model can process any batch size.
    conv1 = Convolution2D(16, 8, 8, border_mode='valid', input_shape=inputshape, subsample=(4, 4), init='glorot_uniform', bias=True)
    model.add(conv1)
    model.add(Activation('relu'))
    model.add(Convolution2D(32, 5, 5, border_mode='valid', subsample=(2, 2), init='glorot_uniform', bias=True))
    model.add(Activation('relu'))
    model.add(Convolution2D(64, 3, 3, border_mode='valid', subsample=(2, 2), init='glorot_uniform', bias=True))
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Activation('relu'))
    model.add(Dense(1024, init='glorot_uniform', bias=True))
    model.add(Dropout(0.5))
    model.add(Activation('relu'))
    model.add(Dense(512, init='glorot_uniform', bias=True))
    model.add(Dropout(0.5))
    model.add(Activation('relu'))
    model.add(Dense(256, init='glorot_uniform', bias=True))
    model.add(Activation('relu'))
    model.add(Dense(1, init='glorot_uniform', bias=True))
    # Print model summary here.
    model.summary()
    # model.compile(optimizer='adam', loss='mse')
    return model
```

Figure 6 CNN Implementation Code

.

The input image to the model has a size of 40 * 160. The first convolutional layer has a patch size of 8 * 8 with 16 output channels and a stride size of 4. Then, the outputs are passed through a rectified linear unit (ReLU) layer. The second convolutional layer has a patch size of 5 * 5, 32 output channels and a stride size of 2. This layer is also followed by a ReLU layer. The third convolutional layer has a patch size of 3 * 3, 64 output channels and a stride size of 2. Then, after flattening, a dropout rate of 20% was applied and another ReLU layer was added. Finally, four fully connected layers with 1024, 512, 256 and 1 neurons were added to output the steering wheel angle. Xavier initialization was used to initialize the weight for all the layers. Using xavier initialization ensures that the scale of initialization is based on the input and output neurons. To compile and train the neural network, the following code was used:

To compile and train the neural network, the following code was used:

```
input_size = (40, 160, 3)
model = cnn(input_size)
plot(model, to_file='model.png', show_shapes=True)
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model.compile(optimizer=adam, loss='mse')
try:
    model.load_weights('model.h5')
except Exception as e:
    print(e)
for i in range(len(files)):
    img_train, img_valid, label_train, label_valid = loadcsv([illegible], files[i], discard[i])
    history = model.fit_generator(getData(img_train, label_train, input_size, batch_size[i], flip_img[i]), \
                                  nb_epoch=epochs[i], verbose=1, samples_per_epoch=len(img_train), \
                                  validation_data=getData(img_valid, label_valid, input_size, batch_size[i], flip_img[i]), \
                                  nb_val_samples=len(img_valid))

with open('model.json', 'w') as f:
    f.write(model.to_json())
model.save_weights('model.h5')
```

Figure 7 CNN Compilation

The Adam optimizer, which is a method for stochastic optimization, provided by Keras, was used to optimize the neural network. And mean squared error between the predicted and labeled angle was used as a loss function. In order to train the network, training data captured 33 from Udacity simulator was used. A figure showing the steering wheel angle distribution of the training data was shown below:

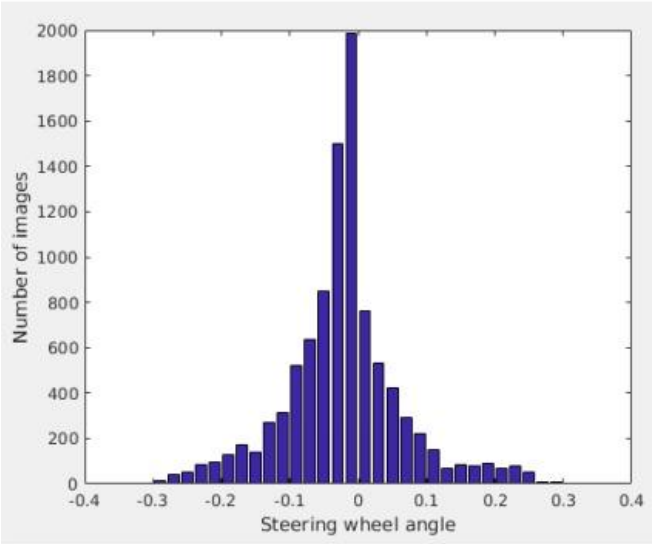


Figure 20 Steering Wheel Angle Distribution

From the distribution, we can tell that in most frames, the car is driving nearly straight, which means that the steering wheel angle is 0. Since there were not enough training data on sharp turns, the car did not perform very well when encountering a very sharp turn. Overall, 30 epochs of training were performed on all the training data with a batch size of 64 and 16 epochs of training were performed on the images with steering wheel angle not equal to 0.

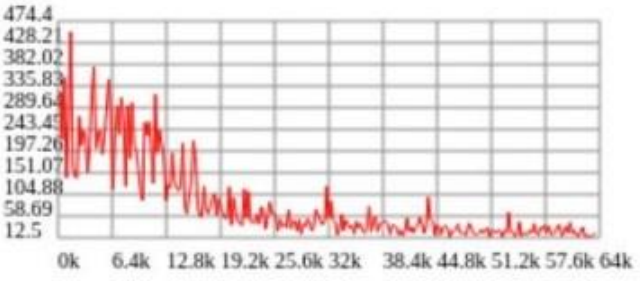
IV. RESULT AND DISCUSSION

In order to evaluate the convolutional neural network, a variety of test was conducted. The first test was done through the use of an online framework called 'DeepTesla' was used. DeepTesla is a online platform for testing end-to-end steering models. The code used in DeepTesla is shown in the following figure:

```
"network" : [
    { "type" : "input", "out_sx" : 200, "out_sy" : 66, "out_depth" : 3 },
    { "type" : "conv", "sx" : 5, "filters" : 16, "stride" : 2, "pad" : 2, "activation" : "relu" },
    { "type" : "pool", "sx" : 3, "stride" : 2},
    { "type" : "conv", "sx" : 3, "filters" : 32, "stride" : 2, "pad" : 2, "activation" : "relu" },
    { "type" : "conv", "sx" : 3, "filters" : 64, "stride" : 2, "pad" : 2, "drop_prob" : 0.2,
      "activation" : "relu" },
    { "type" : "fc", "num_neurons": 1024, "drop_prob": 0.5, "activation" : "relu"},
    { "type" : "fc", "num_neurons": 512, "drop_prob": 0.5, "activation" : "relu"},
    { "type" : "fc", "num_neurons": 256, "activation" : "relu"},
    { "type" : "regression", "num_neurons" : 1 }
],
"trainer" : { "method" : "adadelta", "batch_size" : 32, "l2_decay" : 0.0001 }
```

Figure : CNN Test Code

Since the input images from DeepTesla had different sizes than the training images the team used in the project, the patch size of the first convolutional layer was changed to 5 and a pooling layer with patch size 3 and stride 2 was added. The following plot shows the change of mean squared error as the number of training images fed to the neural network increased:



Mean Squared Errors

As the plot above shows, the mean squared error decayed as the network was fed by more images. After about 57k images were used for training, the error reached a stable value of less than 10. The following figures show visualizations of the images generated by the first and second convolutional layers along with the corresponding training image:

Figure Training image, actual angle and predicted angle

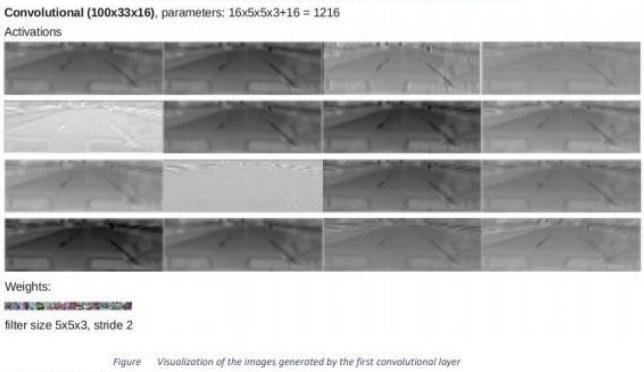
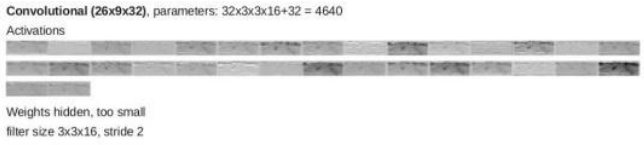


Figure Visualization of the images generated by the first convolutional layer



Convolutional (100x33x16), parameters: 16x5x5x3+16 = 1216

second, and it collects the steering wheel angle of the vehicle at a rate

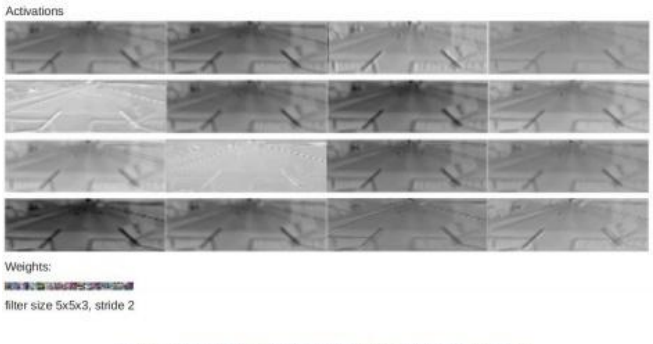


Figure Visualization of the images generated by the second convolutional layer

From the figures above, one can tell that lane markings were automatically extracted as features by the convolutional layers. And the predicted angle was almost the same as the labeled angle. After evaluating the convolutional neural network on Deep Tesla online platform, the team proceeded to do a second test, this time using the Udacity driving simulator. Udacity is a machine learning specific simulation tool with a simple design and environment; Udacity allows users to both test and train CNN models. For this project, Udacity driving simulator was used to determine whether the neural network can successfully keep the car in lane. The overall performance of the CNN model was good. The car was able to drive a whole loop around the simulated track without human intervention. The only problem experienced in the simulation occurred when the vehicle encountered sharp turns; in this case the steering wheel angle predicted by the neural network was sometimes smaller than the actual angle need so the car would drive out of lane for a very brief period time before automatically adjusting back. A video showing the convolutional neural network automatically keeping the car in lane of 30 frames per second. This data can then be used to construct a real-world driving simulator and can also be used as the training data for our neural network. The Convolutional Neural Network model also works as expected, it is able to accurately and correctly produce the correct steering wheel angle for any given image or frame of a video that it is provided with. When provided with many images at a faster than expected rate, the model does struggle and often makes mistakes, however this is a problem that can be fixed with more training epochs as well as some refinement of the code. The simulator portion of the project was begun, with the image calibration portion of the real world 3D simulator completed; however due to time constraints the rest of the real world 3D simulator was not able to be completed. However, the team was still able to test the CNN through the use of the Udacity simulator. Overall, this project has produced a cohesive system of data collection, network model training, and network model testing that can be used to advance WPI's intelligent transportation program.

Camera Calibration [Simulator]

1. Introduction To build a simulator using real world data, we need to be able to transform a 3D world coordinate to 3D camera coordinate, and also transform a 2cameracoordinateto pixel coordinates in the image frame. As a result, it is necessary for us to find the extrinsic matrix and intrinsic matrix for the cameras we used.

2. Intrinsic Matrix In order to map the camera coordinates to the pixel coordinates in the image frame, the intrinsic matrix need to be found. The intrinsic matrix can be expressed as the following:

$$\begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix}$$

In this matrix, fx, fy represents the focal length in pixels. S represents the skew coefficient between x and y axis and cx, cy represents the offsets. To transform a camera based 2D coordinate to 2D point in the image plane, the following formula was used:

$$p_i = K \cdot p_c$$

where pi represents the homogenous pixels on the image plane, pc represents the camerabased 3D coordinate and K represents the intrinsic matrix.

3. Extrinsic Matrix Extrinsic matrix was used to transform a 3D world coordinate to a 3D camera coordinate. The extrinsic matrix can be represented as the following:

$$R_{w,c}\, t_{w,c}$$

Rw,c in the formula represents the rotation matrix of the camera system, and tw, c in the formula represents the translation of the optical center from the origin of the world coordinate. In order to transform a point from world coordinate to camera coordinate, the following formula was used:

$$p_c = \left(R_{w,c}\, t_{w,c}\right) p_w$$

where pc represents the 3D camera coordinate, pc represents the camera-based 3D coordinate, Rw, c represents the rotation matrix and tw, c represents the transformation matrix. Putting the intrinsic matrix and extrinsic matrix together, a 3D world coordinate can be transformed to pixel coordinate on image plane. The following formula was used for this operation:

$$p_i = K \cdot \left(R_{w,c}\, t_{w,c}\right) p_w$$

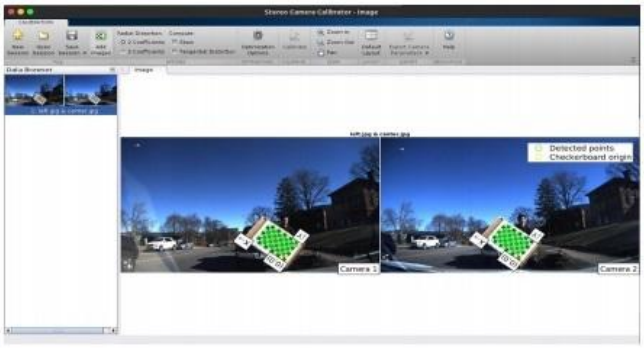
In this formula, Kis a 3 by 3 matrix representing the intrinsic matrix for the camera, (Rw, ctw, c)is a 3 by 4 matrix represe the extrinsic matrix for the camera. Camera calibration was done to 40 find the intrinsic matrix and the extrinsic matrix using the measured image plane coordinates and the world coordinates.

4. Implementation for finding camera parameters

Initially, 'Camera Calibrator' application from computer vision toolbox in Matlab was used to find the camera parameters for individual cameras. A problem for this approach is that since there are small errors when calculating intrinsic and extrinsic matrices, when the camera' locations were calculated based on camera parameters we got, it didn't correspond to the actual location of the cameras. Since the relationships between cameras are already known, these three cameras are treated as two sets of stereo camera, and the 'Stereo Camera Calibrator' application was used. This application takes at least 10 images of checkerboard from a pair of cameras as well as the size of the checkerboard. It will automatically detect the cross points of checkerboard and calculate the camera parameters. Also, the parameters needed to correct radial distortion will also be given. Because the

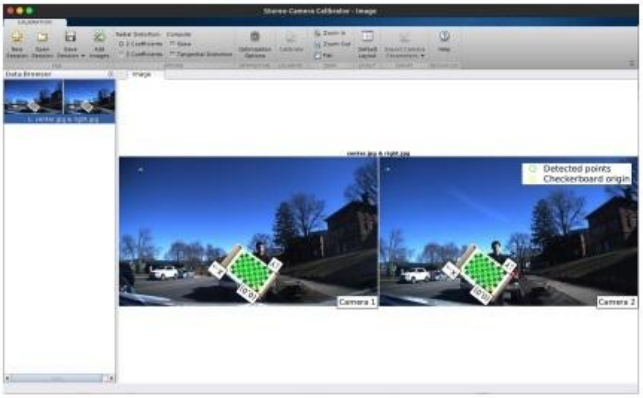
camera locations were now found in groups, the result was more accurate.

The application with an image from left camera and one from center camera was shown in the following:



Calibration Example 1

The application with an image from center camera and one from right camera was shown in the following:



Calibration Example 2

Also, the translation matrices in world units and the rotation matrices of left camera relative to center camera and of right camera relative to center camera were found. Then, only the extrinsic matrices for the center camera need to be found to determine the extrinsic matrices for all three cameras. Initially, we tried to put a checkerboard on the ground and use the application to calculate the extrinsic matrix, but as the figure shown in the following, the checkerboard was too small to be clearly detected.

Calibration Example 3

We then decided to use the endpoints and cross points of lane markings on a parking lot as coordinates in world coordinate. The following image was finally used to calculate the extrinsic matrix of the center camera:

Extrinsic matrix center of camera

The coordinates of the endpoints and cross points were first calculated according to the distances measured. Then, radial distortion was corrected using the camera parameters calculated by the application. The resulting image was shown in the following

Image correction

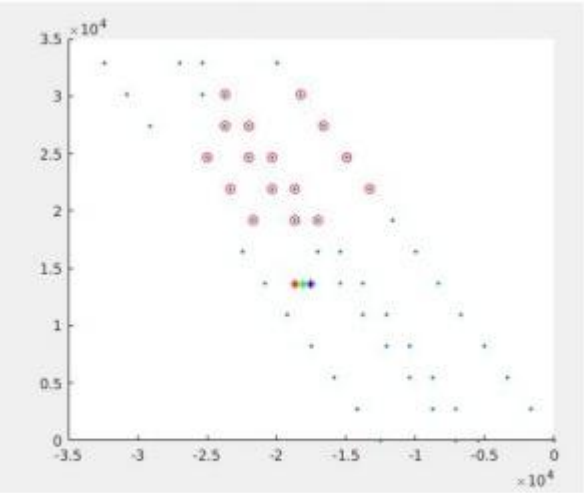
The following code was then used to calculate the extrinsic matrices for the cameras:

```
182 -   [rotationMatrix,translationVector] = extrinsics(img211(:,:),worldPoints(:,:),cameraParams(2));
183 -   cameraEx(2) = [rotationMatrix;translationVector];
184
185 -   cameraEx(1) = cameraEx(2)*r1to0;
186 -   cameraEx(1)(4,:) = cameraEx(1)(4,:) + t1to0;
187
188 -   cameraEx(3) = cameraEx(2)*r1to2;
189 -   cameraEx(3)(4,:) = cameraEx(3)(4,:) + t1to2;
```

Calculatina Extrinsic Matrices

The function ‘extrinsics’ from computer vision toolbox in Matlab was used to get the rotation matrix and translation vector for the center camera. This function takes in the coordinates of the points on the image without lens distortion, the world points we calculated and the camera parameters we got from the application. Then, based on the relationship between left, center and right cameras, all three extrinsic matrices were calculated.

The camera positions in the world coordinate were then plotted to make sure that the extrinsic matrices were accurate. The resulting plot was shown in the following:



Calculated Camera Positions

In this figure, the red circles represent the points we used, the red, green and blue stars represent the left, center and right cameras. The locations of the cameras shown in the figure were very close to the actual locations of the cameras in the chosen image.

V. CONCLUSION

The team succeeds in creating a data collection system and a Convolutional Neural Network (CNN) model for intelligent transportation. The simulator portion proved to be beyond the scope of this paper; however substantial advances in the simulator were made in the form of the camera calibration-progress that can be built upon by future projects. The data collection system produced excellent results, logging the speed, steering wheel angle, and stitching three different camera angles together. The Convolutional Neural Network model is able to produce the correct steering wheel angle for any given image it is provided with. The simulator portion of the project was begun, with the image calibration portion of the real world 3D simulator completed; however, the scope of the real world 3D simulator proved to be too large and was not able to be completed due to time constraints the rest of the real world 3D simulator was not able to be completed. By developing these tools, the team was able to enhance and advance intelligent transportation program. This would result in more efficient and robust data collection, CNN models, and true to life tests.

REFERENCES

1. Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. "End to End Learning for Self-Driving Cars."NVIDIA. NVIDIA Corporation, 25 Apr. 2016. Web. 20 Sept. 2016. . "
2. DeepTesla - Deep Learning for Self-Driving Cars." http://selfdrivingcars.mit.edu/deepteslajs/. Accessed 21 Mar. 2017.
3. Brown, A. (2017, February 22). Udacity/self-driving-car-sim. Retrieved March 22, 2017, from https://github.com/udacity/self-driving-car-sim
4. FLIR Integrated Imaging Solutions. (2017). FlyCapture SDK. Retrieved March 22, 2017, from https://www.ptgrey.com/flycapture-sdk

AUTHORS PROFILE

Mr. Vijay Bhanudas Gujar, M.Tech in Computer Engg, from DBATU lonere. Publish paper in Genetic Algorithm, Big data Analytics. Deep Learning, Machine Learning. Automata Theory, Computer Architecture.