Curvy DeepPicar

Patrick McNamee
Department of Electrical Engineering and Computer Science
University of Kansas
Lawrence, KS
p678m854@ku.edu

***Abstract*—Convolutional Neural Networks (CNNs) are a common implementation for vision based driving on autonomous vehicles. One common technical issue for these CNN vision based systems is that the neural network outputs tend to be very noise with no guarantees of output smoothness as the vehicle traverses the environment. Previous work has shown that using the network outputs as Bèzier curves for an outer loop navigation controller with an inner pursuit curve controller leads to smooth performances and outperform other network architectures. This work follows a natural extension implements a CNN with Bèzier curve outputs as part of the controller module rather navigational module on a physical platform for vision navigation and evaluates the results against a standard CNN as a direct controller. Both evaluation of final training results and a physical experiment indicate that implementing Bèzier CNNs as either direct controllers or an outer loop navigation controller would improve the performance of autonomous automobiles for lane navigation.**

***Index Terms*—machine learning, autonomous vehicles, embedded platforms, guidance navigation and control, convolutional neural networks**

I. Introduction

Modern car manufacturers are developing autonomous automobiles for future use in both civilian and military applications. There are various complex environmental aspects that are required for consideration in order for autonomous agents to succeed in driving tasks such as sensors for object detection and localization, path planning, and state prediction for object avoidance [1]. These aspects traditionally rely on computationally complex and resource intensive algorithms that may be too slow to implement on a real-time system. An alternative is to use neural networks which are universal approximators [2] to approximate the algorithms for a loss of functional accuracy but an increase in computational throughput. As such, neural networks have been frequently used for autonomous vehicle.

Usage of neural networks in autonomous ground vehicles has been around for decades with ALVINN being one of the first examples of vision-based navigation [3]. ALVINN used a 32 by 32 video input along with a 8 by 32 range finder input feed into a hidden layer of 29 units before the output layer of 46 units. The wheel steering angle was chosen based on the values of the 46 output units and ALVINN was quite successful at its various driving tasks. Modern implementations of autonomous cars, like the one developed by NVIDIA, are empowered by modern computational performance have since moved from the relatively simple network architecture used in ALVINN and currently use CNN which includes convolutional layers as well as significantly larger layers size [4]. While these larger neural networks have allowed for more complex behavior of the autonomous agent, there are still issues that result from the nature of neural networks. Often the outputs of neural networks are rough in the sense small manipulations on the input layer can produce drastically different outputs and there are no guarantees of output behavior in unexplored environments. Still, CNNs are potential methods to work towards fully autonomous driving and there are several implementation techniques to include them.

Accompanying GitHub Repo: https://github.com/p678m854/EECS-753-Project

A. Background

There are various ways for CNNs to be implemented for autonomous vision-based automobile navigation as reference in Fig. 1. The most direct method is to have the CNN take images and translate them into direct control inputs such as the steering angle or throttle as in [5]. While direct, this implementation is limited in that the controller is limited by the input rate of the camera systems as well as the network throughput implementation. It is also susceptible to network output roughness which can lead to high control jerk and unnecessarily fatigued physical components. This implementation of a direct controller is referred to in the rest of this work as image-to-point for translating a camera image into a point in the control input space.

Two alternative implementations that are indirect schemes are navigational implementations demonstrated in DeepRacing [6]. Rather than directly control the car, the CNN outputs desired positions for navigate with a lower level control scheme closing the loop. There are two different ways to represent navigational points using discrete and continuous methods. A discrete method is simply to transform the input into a collection of ordered waypoints for navigation. However, a CNN with rough outputs will still produce noisy outputs that may cause unwanted behavior. Additionally as the number of waypoints changes, the network architecture needs to be adjusted.

Rather than represent the waypoints or as discrete points, the waypoints can be represented as a continuous polynomial curves. Bèzier curves $\mathbf{B}$ are useful representation as any polynomial of degree d can be implemented with $d+1$ points $\mathbf{P}_k$ in k dimensional space as poles by (1) using a parameter t.

$$\mathbf{B}(t) = \sum_{k=0}^{d} \binom{d}{k} (1-t)^{d-k} t^k \mathbf{P}_k \quad t \in [0,1] \tag{1}$$

This representation is advantageous as it can represent the same time window as the discrete waypoint representation but has an infinite number of points so the number of waypoints generated is not upper bounded. Additionally the waypoints are guaranteed to be smooth since they are on a polynomial curve and experimentally DeepRacing showed that this Bèzier representation was the best performing network for CNNs in the simulated racing domain [6]. While previous results are from simulations, to the authors knowledge this has not been verified any physical platform.

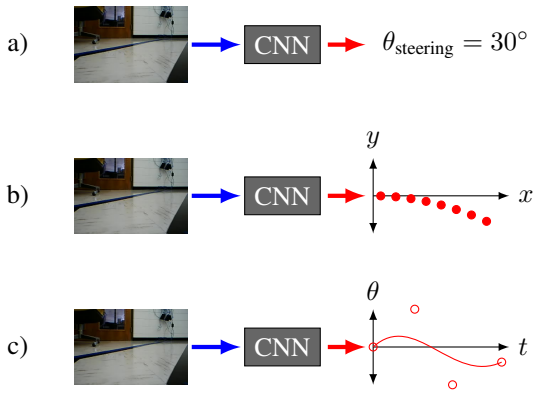

Fig. 1. Various Vision-Based Autonomous Steering Implementations. From top to bottom; a) image to direct control command, b) image to localized waypoints (●), and c) image to curve using poles (○).

While most work is focused on personal or racing automobiles, there have been implementations of vision-based navigation on small, radio-controlled cars. These vehicles are significantly cost-effective as research platforms and testing embedded systems. Previous work at the University of Kansas has demonstrated implementing the NIVIDIA DAVE-2 neural network architecture on a Raspberry Pi for a CNN image-to-point vision navigation system on a physical platform referred to as DeepPicar [5]. From a cost standpoint, the DeepPicar platform is ideal for testing a real implementation of a CNN with Bèzier curve output although due to hardware and sensor limitations, an image-to-curve controller with the curve in the control input space will be used to compare the Bèzier implementation of a controller against a standard image-to-point CNN controller rather than having the Bèzier CNN be the navigational controller as in [6]. This is to avoid any heavy computation loads due to simultaneous localization and mapping (SLAM) algorithms which are required for positional estimates on DeepPicar.

B. Motivations

Previous work has shown the use of CNNs outputting Bèzier curves for navigation problems in simulated racing outperforms other techniques such as a CNN outputting navigation waypoints or direct control networks [6]. However the simulation relies on position estimates which may not be feasible onboard small platforms or in environments with no direct position estimates. Additionally, the Bèzier CNNs have, to the knowledge of the author, not been tested on a physical platform as of this work. Hence this work seeks to implement the a CNN outputting Bèzier curves in a direct control scheme for navigating a track on DeepPicar to validate findings from DeepRacing and extend the usage of Bèzier curves to solve control problems in addition to previous investigated navigation problems.

II. Novel Contributions

This work will implement a CNN outputting Bèzier control curves for an embedded autonomous automobile platform for course navigation which has not been tested before on a physical platform or as a direct control command. This new implementation will be tested against a standard CNN with direct control output as used in previous work [5] on the same physical platform.

III. Convolutional Neural Networks

The CNNs are based off of the DAVE-2 CNN architecture from NIVIDIA [4] and is shown in Fig. 2. Previous work has been implemented on a Raspberry Pi 3B+ for the DeepPicar [5] and this work will use the same implementation as [5], albeit on a Raspberry Pi 3A+, to form a direct comparison with previous implementations. The input to the CNNs is a Playstation Eye Camera which generates 320×240 RGB image frames which are resized to 200×66 to match the DAVE-2 inputs. Inside the CNNs, the image is normalized with default Tensorflow 2.x layer parameter values across the image width, height, and channels before passing though multiple convolution layers. Layers 2, 3, and 4 use 5×5 kernels with a width and height stride of $(2, 2)$ while layers 5 and 6 use 3×3 kernels with a width and height stride of $(1, 1)$. After layer 6, the network is flattened to dense layers whose layer width is reduced until a single output unit. The activation functions used for layers 2 through 10 are rectilinear linear units (ReLU) while the last layer uses a hyperbolic tangent function. These activation functions are consistent with implementations in previous work [4], [5] but the use of ReLU for the hidden layers are consistent with proofs that neural networks are universal approximators [7] and the hyperbolic tangent allows for outputs to be bounded to match the automobiles steering wheel angle limits $[\theta_{\min},\ \theta_{\max}]$ by a linear transformation.

A. Training Datasets

There are two publicly datasets available, and investigated, to train radio-control autonomous cars for tracks. The first dataset is a popular, non-academic, but relatively small and non-timestamped dataset consisting of only 219 example images but has a continuous steering angle ranging [8] while the other dataset is the DeepPicar dataset which consists of 11,000 timestamped example images but only has a collection of discrete wheel angles in the set $\{-30°,\ 0°,\ 30°\}$ degrees

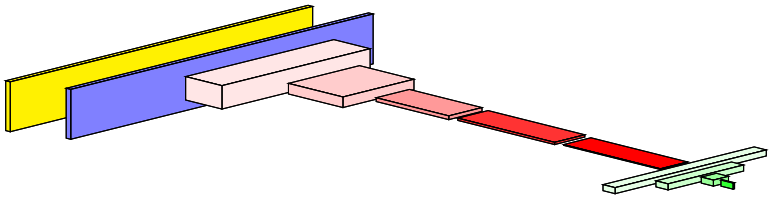

Fig. 2. Visualization of DAVE-2 Neural Network [4]. Yellow is an input image, blue is the regularization layer, red is a convolution layer, and green is a dense layer. Dense layers are exaggerated for viewing and flattening omitted.

TABLE I
IMAGE-TO-POINT CNN MODEL USING DOMAIN AUGMENTATION (ACC: 38.71%)

		Model Predicted		
		Left	***Center***	***Right***
Record	***Left***	141	1,147	28
	Center	820	3,334	306
	Right	850	3,591	783

but uses the same physical platform as this work [5]. As the Bèzier curves deal with continuous outputs, it would be more advantageous from a training perspective to attempt to use Data Augmentation to train the various CNN models on the first dataset and then evaluate the CNN models on the larger second dataset to see if any trained model can be transferred to the matching physical platform. For the training on [8], all images were loaded into memory and an application of a horizontal image flight was applied to augment the dataset. The target steering wheel angles were scaled so that the models was only attempting to output in the range $[-1,\ 1]$ where -1 and 1 indicate a steering wheel angle of $-30°$ degrees and $30°$ i.e. $30°$ left and right respectively. For train the CNN whose architecture is in Table III, a 80-20 test-validation set split was used with a mean squared error (MSE) loss function and the model was trained for 100 epochs using the ADAM optimizer with parameters in Table IV.

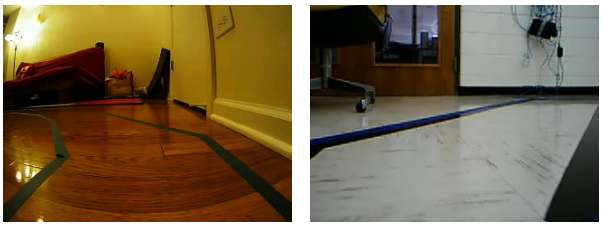

Fig. 3. Examples from the datasets where left is from [8] and right is from [5].

To analyze the performance of the model in the new domain of [5] which has a discrete controller, the CNN model was used to predict the continuous wheel angle for all 1,000 frames of each of the 11 videos and the output was rounded to the closes steering angle in $\{-30°,\ 0°,\ 30°\}$. A cross-correlation matrix of the model outputs versus the record steering wheel angles are displayed in Table I. The accuracy was not deemed high enough to use the model in the domain of [5] i.e. the DeepPicar platform so data augmentation cannot be used in this work. However, the [5] dataset outputs still need to be modified to appear to be continuous. To achieve this effect, the CNN models will train on a central moving average filter of the steering wheel angles with a filter window of 5 data points which are nominally spaced at 50 milliseconds.

TABLE II
IMAGE-TO-POINT CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

Layer	**Layer Type**	**Dimension**	**Parameters**
1	Normalizer	$66 \times 200 \times 3$	79,200
2	2D Convolution (5×5)	$31 \times 98 \times 24$	1,824
3	2D Convolution (5×5)	$14 \times 47 \times 36$	21,636
4	2D Convolution (5×5)	$5 \times 22 \times 48$	43,248
5	2D Convolution (3×3)	$3 \times 20 \times 64$	43,248
6	2D Convolution (3×3)	$1 \times 18 \times 64$	43,248
7	Flattening	1152	0
8	Dense	100	115,300
9	Dense	50	5,050
10	Dense	10	510
11	Dense	1	11
		Non-trainable	79,200
		Trainable	252,219
		Total	331,419

B. Image-to-Point CNN

The image-to-point CNN is a recreation of the models trained in previous work [4], [5] with the various layers types, dimensions, and total parameters listed in Table II. There is a discrepancy between the original DeepPicar previous work [5] and [4] where the first normalization layer is excluded from the CNN. Since the original DeepPicar was using the DAVE-2 CNN as the reference model, this work will follow the DAVE-2 model more strictly and include the layer normalization. For training, the CNN will use the mean squared error (MSE) loss function with the recorded wheel angles being proportionally scaled to the range $[1, -1]$. Afterwards, the model will be implemented onto the DeepPicar as an image-to-point navigation controller operating at 20 Hz.

C. Image-to-Curve CNN

The image-to-curve CNN cannot be the same network architecture as the image-to-point architecture as the outputs of the CNNs are fundamentally different. Image-to-point controllers have a one-to-one mapping so if the image is being taken from the camera for navigation at a fixed rater, the wheel angle will also be scheduled to update at the same fixed rate, in this work 20 Hz. However, the Bèzier CNN operates with a time window for the paramter t to be in the range $[0,\ 1]$ so the mapping can

TABLE III
BÈZIER CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE

Layer	Layer Type	Dimension	Parameters
1	Normalizer	$66 \times 200 \times 3$	79,200
2	2D Convolution (5×5)	$31 \times 98 \times 24$	1,824
3	2D Convolution (5×5)	$14 \times 47 \times 36$	21,636
4	2D Convolution (5×5)	$5 \times 22 \times 48$	43,248
5	2D Convolution (3×3)	$3 \times 20 \times 64$	43,248
6	2D Convolution (3×3)	$1 \times 18 \times 64$	43,248
7	Flattening	1152	0
8	Dense	100	115,300
9	Dense	50	5,050
10	Dense	4	204
11	Reshape	4×1	0
		Non-trainable	79,200
		Trainable	251,902
		Total	331,102

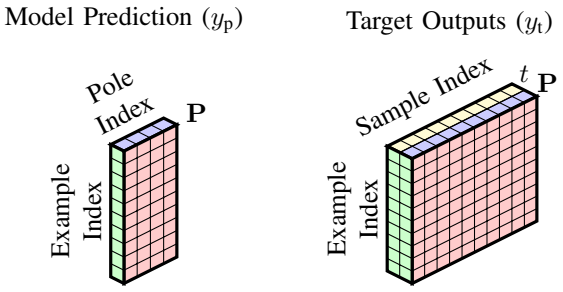

Fig. 4. Model Prediction and Target Outputs

be one-to-many i.e. the Bèzier CNN neural network operating at a slower rate than the wheel angle controller and trains to fit the Bèzier curve to all points in the time window. For this work, time window selected for the t parameter was 0.5 seconds but it is possible for the time windows to overlap. The Bèzier CNN will always predict for the full time window but the overall navigation could update the pole point values say every 0.25 seconds. If there is no time window overlap than the image-to-curve CNN operates as an outer loop controller at 2 Hz with the a wheel angle controller updating at 20 Hz which means there is 10:1 ratio between wheel angles and camera frames assuming both controllers are real-time tasks. This reduction in required CNN task rate reduces the computational load for the embedded system as the CNN frame processing occurs less frequently to the image-to-point models but this reduction depends on the amount overlap between subsequent time windows. If there is some overlap then the 10:1 ratio can be reduced up to a 1:1 ratio as with the image-to-point CNN i.e. the Bèzier CNN is being used to generate new poles at a 20 Hz rate. In the extreme 1:1 case, the Bèzier CNN can simply use the first, i.e. 0 indexed pole, to use as the steering wheel angle in an image-to-point control scheme which may yield different results than just the standard image-to-point CNN controller as the Bèzier CNN trains with future steering angles. Two overlap ratios will be tested, a 10:1 and 1:1 ratio to see the overall behavior.

Training the Bèzier CNN is different from standard neural networks as the model predictions and the target outputs for training are differently shaped tensors as shown in Fig. 4. Where the image-to-point CNN was only responsible for the steering wheel angle $\theta_{\text{steering}} \in \mathbb{R}^1$, the image-to-curve CNN outputs poles $\mathbf{P}_k$ to form the Bèzier curve hence the CNN output needs to exist in $\mathbb{R}^{(d+1)\times 1}$ space. To accomplish this, the last two layers are changed from the image-to-point CNN to the image-to-curve CNN with the polynomial output being a cubic with the resulting network architecture being shown in Table III. For the target outputs during the training session, both the point and associated non-dimensional parameter t value must be recorded. This leads leads to a mismatch in dimension size where, assuming the first tensor dimension is the example index, the second dimension of the tensors do not match. The model output has the second dimension reserved for indexing the poles forming the curve while the target output uses the second dimension to index the sample recorded points to fit the curve to.

Previous work rectified this dimensional discrepancy by generated output points $\mathbf{B}$ using a matrix representation $\mathbf{B} = \mathbf{A}(\mathbf{t}, d)\mathbf{P}$ where $\mathbf{A}(\mathbf{t}, d)$ is the matrix whose elements are $(1-t)^{d-k}t^k\binom{d}{k}$ from (1) determining points by the matrix of output poles $\mathbf{P}$ and the corresponding parameter vector $\mathbf{t}$ [6]. While this implementation is simple, readily converted to a graph structure for GPU training, and easily integratable with preexisting loss functions it overly constrains the outputs as $\mathbf{A}$ needs to be determined a priori so the sampling can only happen at predetermined points. This work instead uses a custom implementation with functional mapping in Tensorflow 2.x to implement (2) which is sum of squared errors for arbitrary n dimensional point spaces. As the implementation is a functional mapping, it is a more flexible approach that is robust to unique time window sampling spaces as well as ragged tensors i.e. the number of samples in a time window could potentially be different. A visual comparison of the model predicted tensors and the target outputs for applying (2) is shown in Fig. 4 and the recorded steering wheel angle values have been been mapped to $[-1,\ 1]$ such that the extreme angles are $\pm 45^\circ$ degree turns so that the poles can allow for an intermediate point in the time window to achieve the vehicle limit of $\pm 30^\circ$ degree turns starting from a non-extreme value.

$$\mathcal{L}_{\mathbf{B}}(y_p,\ y_t) = \sum_e \sum_s \Big\| y_t(e,s,1:) - \sum_{k=0}^{d} (1 - y_t(e,s,0))^{d-k} y_t(e,s,0)^k y_p(e,k,:) \Big\|_2^2 \quad (2)$$

D. Training

The dataset is randomly split into a train set of 9 videos and a test set of 2 videos. Each of the videos have output keys generated in previous work where each video frame is matched to a recorded steering wheel angle. Since loading in all of the videos at once is resource expensive as the available hardware for this work only has roughly 2 Gigabytes of RAM available for the holding the input and output tensors, an

TABLE IV
TRAINING PARAMETERS

Parameter	Value
Training Videos	[2, 9, 7, 8, 10, 1, 11, 6, 4]
Loops over Training Set	4
Test Videos	3 and 5
Epochs/200 Frames	50
Batch Size	256
Optimizer	ADAM
Learning Rate	$\alpha = 0.001$
Decay Rate	$\beta_1 = 0.9$, $\beta_2 = 0.999$

Fig. 5. Training history of standard CNN (top) and Bèzier CNN (bottom).

unusual training scheme is performed. Rather than training over all frames of the 9 videos in batches as is traditional, the models are trained on a video as specified by sequentially iterating over an ordered list of videos before looping back and training on the list again. For training on a video, 200 frames are loaded into memory with a horizontal flip augmentation to yield 400 training examples for the model to train on. After training, the next 200 frames are loaded in training and this repeats until the model has trained on all 1,000 frames of a video. The specific training parameters and listings are shown in Table IV and the training histories are shown in Fig. 5. The training history shows periodic behavior which generally does not appear in traditional training schemes but overall loss function seems to be reducing as the training persists with minimal reductions towards the end of training. This indicates that the CNN models have not been overfitted and may proceed to be evaluated.

TABLE V
IMAGE-TO-POINT CNN MODEL (ACC: 20.45%)

		Model Predicted		
		Left	***Center***	***Right***
Record	***Left***	48	31	21
	Center	156	156	188
	Right	54	136	205

TABLE VI
IMAGE-TO-CURVE BÈZIER MODEL (ACC: 38.10%)

		Model Predicted		
		Left	***Center***	***Right***
Record	***Left***	83	46	33
	Center	268	343	313
	Right	177	268	336

IV. EXPERIMENTS

There are two methods to analyze the quality of the CNN; a quantitative accuracy check using the test videos and a qualitative check using a navigational track. These two methods are consistent with previous work [5] and allows for a fair comparison of the models in this work.

A. Model Accuracy

Accuracy to the record values of the steering wheel is used to quantitatively evaluate the model quality for the various approaches. Accuracy is determined by iterating through all 2,000 total frames of the Video 3 and 5 of the test set in [5] and comparing continuous CNN outputs to their closest correspondence in the discrete $\{-30°, 0°, 30°\}$ degree viable wheel angles with results shown in Tables V, VI, and VII. The comparison previous work [5] using the same CNN architecture as the image-to-point network has shown accuracy of approximately 70% although those models do not have normalization layer which is thought to be the reason why the baseline accuracy CNN are so poor in comparison. However, the results do show that using Bèzier curves in the model networks do drastically improve the performance as they match the truth values at higher accuracy. Interestingly, reducing the ratio of control points to image frames down to a 1:1 drastically improved the CNN controller which indicates that using future steering angles greatly helps in improving the one-off prediction accuracy of image-to-point CNN controllers. Videos involving the overlay of the predictions are in the accompanying GitHub repository for validation in the video quality check folder of the reports subdirectory.

TABLE VII
IMAGE-TO-POINT BÈZIER MODEL (ACC: 66.75%)

		Model Predicted		
		Left	***Center***	***Right***
Record	***Left***	60	111	9
	Center	48	667	261
	Right	13	223	608

B. Physical Driving

While the models were trained int Tensorflow 2.x, the DeepPicar uses a Raspberry Pi specifically a Raspberry Pi 3A+ in this work. This platform does resource limitations and to the best effort of this work cannot install any Tensorflow modules after Tensorflow 1.11 and anyone trying to replicate this work needs to be wary of the size of various Python modules being installed. There are known issues with using Python 3.7 and onwards with Tensorflow 1.x on the Raspberry Pis and some dependencies modules from Tensorflow will cause issues during installation. The associated GitHub Repo of this work will got into more details on the software setup for the DeepPicar. There is a slight modification of implementation of the CNNs for backwards compatibility to Tensorflow 1.11. The first layer of all the CNNs trained in this work was changed from a normalization layer to a batch normalization layer which for batch sizes of one image during deployment are nearly identical. The only difference is that the batch normalization layer used a moving average of the mean and standard deviations during inferences but considering the time frame of the experiments will be roughly 50 seconds, this was deemed acceptable as it was still a reasonably close layer behavior match.

The navigational track being used for this work is consists of straight line segments connected at 45° degree angles with the track being approximately 50.4 centimeters in width. The dimensions of the line segments are shown in Fig. 6 for reproduction purposes but the track occurs in a residential home with wood and vinyl tiles in dim to moderate lighting. Each of the models and schemes were used to attempt a single navigation of the track with the default implementation of [5] of 50% throttle with 2 CPUs being used for the CNN inference. Since all models were initially trained on a different dataset, none of the schemes used to successfully navigate the track. The most successful scheme was the Bèzier image-to-point (1:1 ratio of images to steering wheel angles) which managed to autonomously turn on the 287-45.7-86.4 joining segments. Otherwise both the Bèzier image-to-point and traditional image-to-point had strong tendencies to drive straight with human intervention required for most of the turns. The least successful model was the Bèzier image-to-curve model which was only turning right but this is regarded as an issue with the different track environment from training rather than a general performance behavior. For future repeatability, the associated scripts for implementing this work as required for DeepPicar by its previous work [5] are included in the associated GitHub repo in the deeppicar-scripts subdirectory. Both the first person view of the DeepPicar driving as well as an exterior view are also given in the GitHub repo in the deeppicar-testing folder of the reports subdirectory.

V. Results and Conclusions

This work illustrated that the Bèzier CNNs can be used to generate smoother and more predictable behavior than traditional CNNs in image-to-point controllers. While the use of Bèzier output CNNs as image-to-point controllers are consistently better than standard CNN implementations, the image-to-point Bèzier CNN was only performed better than the standard CNN when analyzing the results of the training and was not experimentally shown. Although reserved the results suggest that Bèzier CNNs could yield performance increases and output smoothness if integrated into currently developing autonomous vehicles.

VI. Future Work

The training dataset [5] is initially limiting; wheel angles are only three variations and there is a one-to-one correspondence between wheel angles and video frames. Curve fitting to noisy data best works with more sampling points so it would preferable to have a one-to-many mapping between image frames and recorded wheel angles. Rather than constructing the dataset by forming corresponding pairs of images and wheel angles, there could two threads of data collection with one thread for the camera at a nominal operational rate and the second thread is for recording the wheel steering angle at a faster but sustainable sensor rate. Alternatively the original [5] could be modified by estimating intermediate point values to upsample the presented data from 20 Hz to an artificially faster sensor rate.

Aside from the model training improvements, the physical platform testing could be improved. Rather than testing in an entirely new environment, it would be better to create another track at the original DeepPicar track location to match the environment. This will help to give a more unbiased assessment of the models and controller schemes behavior.

References

[1] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 287–296.

[2] Z. Wang, A. Albarghouthi, G. Prakriya, and S. Jha, "Interval universal approximation for neural networks," 2021.

[3] D. A. Pomerieau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie Mellon University, Tech. Rep., January 1989.

[4] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[5] M. G. Bechtel, E. Mcellhiney, M. Kim, and H. Yun, "Deeppicar: A low-cost deep neural network-based autonomous car," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018, pp. 11–21.

[6] V. S. B. Trent Weiss and M. Behl, "Deepracing ai: Agile trajectory synthesis for autonomous racing," in *International Conference on Intelligent Robotis and Systems (IROS): Workshop on Perception, Learning, and Control for Autonomous Agile Vehicles.* IEEE Robotics and Automation Society, 2020.

[7] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[8] D. Tian, "Deeppicar," Online: https://github.com/dctian/DeepPiCar, March 2019.

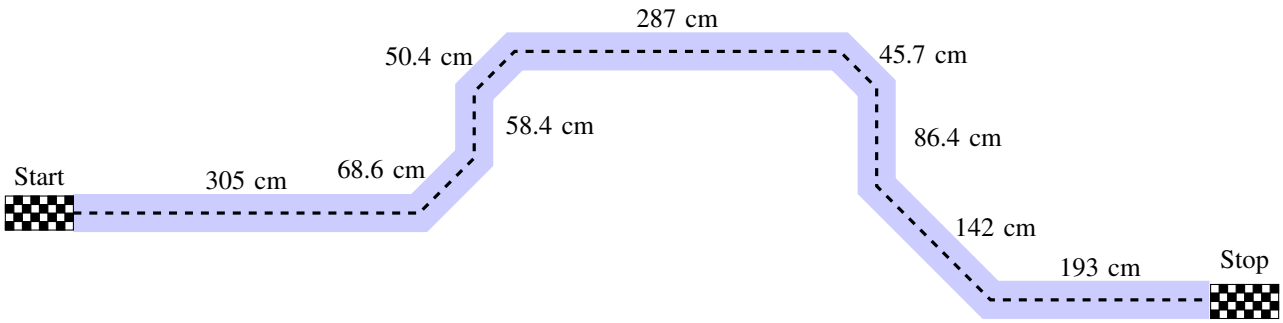

Fig. 6. Navigational Track.