2022-01-0369 Published 29 Mar 2022

# Virtual Evaluation of Deep Learning Techniques for Vision-Based Trajectory Tracking

**Ameya Salvi, Jake Buzhardt, Phanindra Tallapragda, and Venkat N Krovi** Clemson University

**Jonathon M. Smereka and Mark Brudnak** US Army DEVCOM-GVSC

***Citation:*** Salvi, A., Buzhardt, J., Tallapragda, P., Krovi, V.N. et al., "Virtual Evaluation of Deep Learning Techniques for Vision-Based Trajectory Tracking," SAE Technical Paper 2022-01-0369, 2022, doi:10.4271/2022-01-0369.

Received: 14 Feb 2022 Revised: 14 Feb 2022 Accepted: 12 Jan 2022

## Abstract

Artificial intelligence (AI) enhanced control system deployments are emerging as a viable substitute to more traditional control system. In particular, deep learning techniques offer an alternate approach to tune the ever increasing sets of control system parameters to extract performance. However, the systematic verification and validation (to establish the reliability and robustness) of deep learning based controllers in actual deployments remains a challenge. This is exacerbated by the need to evaluate and optimize control systems embedded within an operational environment (with its own sets of additional unknown or uncertain parameters). Existing literature comparisons of deep learning against traditional controllers, where they may exist, do not offer structured approaches to comparative performance evaluation and improvement. It is also crucial to develop a standardized controlled test environment within which various controllers are evaluated against a common metric. Hence, in this paper, we evaluate deep learning based controllers by a structured selection of pantheon of evaluation metrics and employ the insights to propose further modifications to the deep learning algorithms. We first evaluate a high resolution proportional-integral-derivative (PID) controller to serve as a common benchmark and develop the suite of evaluation metrics for a mobile robot to create a point-to-point trajectory within in the simulation environment. Against this backdrop, we then evaluate alternate variants of imitation learning and deep reinforcement learning controllers which use camera image stream as an input to develop or tune trajectory tracking output parameters. Subsequent deployments of control strategies on actual hardware (in a Sim2Real transition) is anticipated, where the developed evaluation metrics can be used as a validation tool when deploying the control strategy on actual hardware.

## Introduction

The past decade has seen a resurgence of interest in deployment of AI based (and more specifically) deep learning (DL) based control techniques. Such control techniques leverage iterative training regimes to learn end-to-end control schemes for different vehicle and robotic control applications. The concurrent revolutions in computing (multicore and GPGPU based systems), communication (Wi-Fi, Bluetooth), enhanced modeling, simulation and test stand tools leading to greater data availability (large-scale training sets) and ease of access to end-to-end web-based frameworks (Keras, Tensorflow) have each contributed to the increased deployments [8]. The early successes then have set the stage for ever-increasing complexity of deployments in actual real-world settings, than could have even been contemplated even a decade ago.

As opposed to traditional control schemes, the end-to-end learning nature of these data-driven controllers offer tantalizing possibilities in realizing the end-to-end relations between the observations and control actions [1]. They can take advantage of either increased simulation fidelity and/or explicit real-time data-streaming or hybrid combinations of the two in X-i-L {X=software, hardware, human} settings. It is also noteworthy that the principal challenges with DL control systems comes in the training process. In actual deployments, the deep neural networks based control systems actually has lower real-time computation needs compared to a traditional feedback controller [9].

Yet, despite all these potential benefits, there are significant challenges to deployment of DL based controllers that need to be systematically addressed. First and foremost, is a better characterization of the size, scope and nature of the control deployments i.e. what can they not do? And what can they do? And what can they replace? Second, verification and validation remain an open challenge given the black box nature of these controllers. The lack of structured approaches to understand the internal working of such controllers inhibits ways to target regions of suboptimal performance and improve only those specific regions. Third, the existence of a support ecosystem coupled with a workflow for training, test, deployment and verification (ideally within a continuous improvement cycle) would be crucial. Given the limited scope of this

manuscript, we will restrict our attention primarily to the first two challenges in some depth and with tangential attention paid to the third.

To overcome these challenges, key performance indices or evaluation metrics are required that can help compare multiple data-driven controllers and traditional controllers through the same lens. The evaluation metrics should be independent of the inner workings of the controller and should be derived only from the input-output states. It also becomes essential, that these evaluation metrics not only compare performance of two controllers, but also provide insights on the areas where the controller needs to improve. The independence of these metrics from the inner workings of the controller, facilitates comparing any type of feedback controller and provides ways to compare contemporary control techniques against traditional feedback controller.

In this paper, multiple evaluation metrics leveraging the knowledge of standard statistical techniques have been implemented for a particular application of developing learning algorithms for vision-based trajectory tracking. The simplified problem is setup with a skid-steered robot (a SUMMIT-XL robot as shown in figure 1 [11]) performing path tracking operation.

Skid-steered vehicles find lot applications with agriculture, military and in general any off-road terrain where high maneuverability is desired. The interest of this work in skid-steered vehicles lies in the fact that the study of cornering performance, generalized key performance indicators and most of the research is still tailored around bicycle models, thus creating a void in literature for skid-steered vehicles.

This visual servoing task has been set up in a standardized simulation environment of robotics simulation software CoppeliaSim [11] and using a remote API client to interface with MATLAB [7], where the learning algorithms are developed. Such a framework provides with necessary visual fidelity while leveraging the rapid prototyping tools for control algorithms provided by MATLAB.

In this paper, a PID tracking algorithm for path tracking has been first setup in the CoppeliaSim-MATLAB framework. This algorithm helps to identify the input and output values that would be available for parametrizing the evaluation

**FIGURE 1** Robotnik SUMMIT-XL Mobile Robot in CoppeliaSim [11]

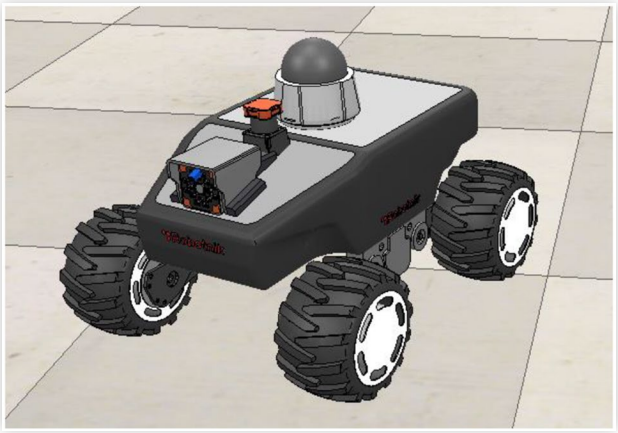
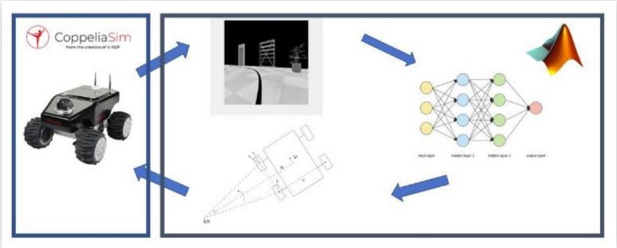


**FIGURE 2** Interfacing MATLAB [7] and CoppeliaSim [11] using remote API




metrics which in turn would be used to benchmark this PID tracking algorithm. These benchmarked values are used to investigate the performance of data driven tracking algorithms like imitation learning and deep reinforcement learning for the same visual servoing application.

# Formulation

## Simulation Setup

**Track Layout and Sensor Information** A vision sensor plugin enables capture of an image of what the robot sees in real time and transfers it to the MATLAB environment. All the image processing steps (as referenced in figure 6) take place in the MATLAB environment. The sensor and the actuation information is communicated between MATLAB and CoppeliaSim at a data transfer rate of 100 Hz.

The objective for track design was to have adequate number of sharp turns and have them in both clockwise and counterclockwise manner. This was to make sure that the training data remains unbiased and the learning algorithms can generalize well on both the directions. The sharpest turn was made to be more than the minimum turning radius of the robot. Skid-steer robots have the ability to turn on the spot like differential drive robots, but this was avoided to reduce the training complexity. The minimum turning radius ($R_{min}$) was found by giving the outer wheels maximum possible positive velocity and the inner wheels minimum possible positive velocity.

**FIGURE 3** Raw camera feed as viewed by the robot in the simulator

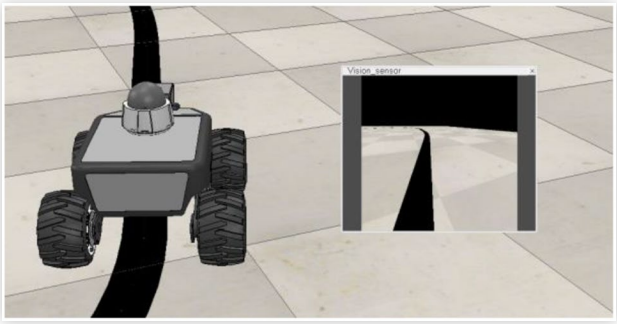


**FIGURE 4** Track layout in the CoppeliaSim software

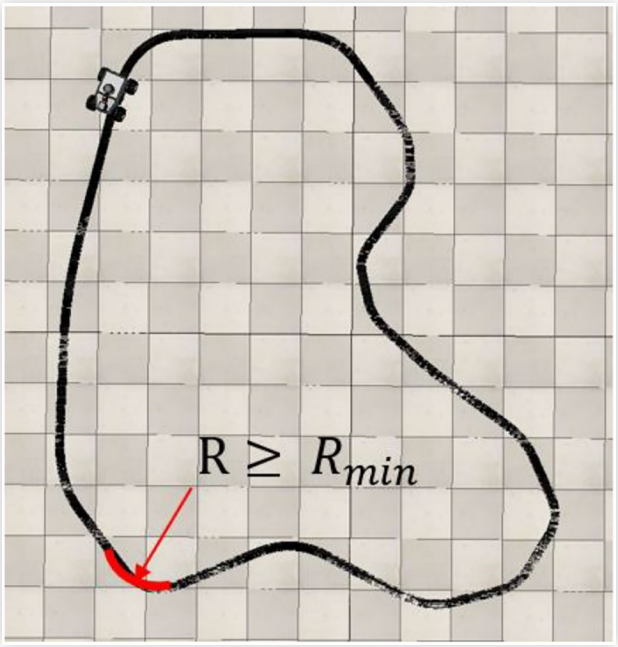
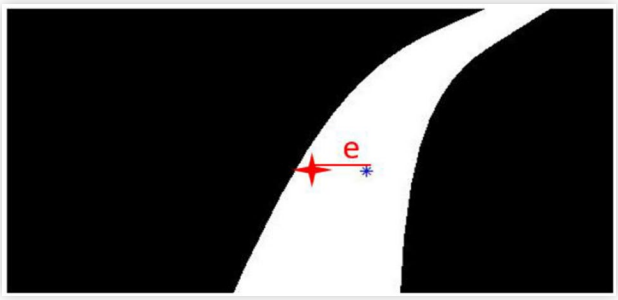



**FIGURE 5** Error generated by calculating the centroid of the high intensity pixels to be used for the feedback controller




$$V = \frac{V_{outer,max} + V_{inner,min}}{2} \quad (1)$$

$\omega$ is acquired from the simulation environment and helps calculate $R_{min}$ as

$$R_{\min} = \frac{V}{\omega} \quad (2)$$

**Proportional Integral Derivative Tracking Scheme** A PID controller is a feedback controller that regulates control actions to maintain a desired set point. In this context, the SUMMIT-XL robot sees the lane and calculates the center of the lane leveraging the image processing tools. The horizontal distance between this lane center and the center of the vision sensor is termed as *error*, which the PID controller tries to minimize. With the lane image as an input, the controller generates a yaw rate for the robot, which controls the steering action. Figure 5 shows the lateral error (e) between the camera center (red star) and the path centroid (blue star).

**FIGURE 6** An end-to-end process flow chart. Steps in the red dotted square get replaced by a neural network for data driven control scheme

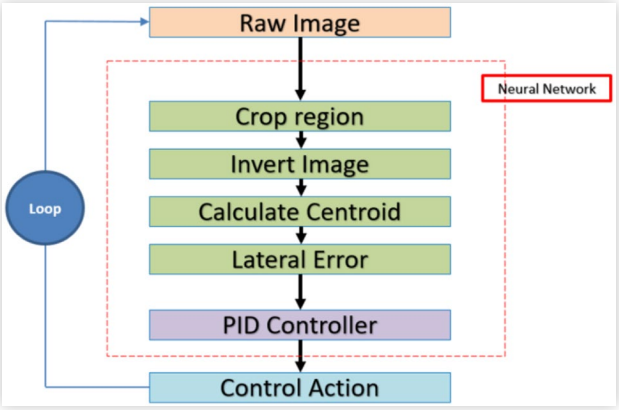



Figure 6 shows the process flow chart of the steps involved when going from a raw camera image to a control action. The steps inside the dotted line are the ones that can be directly replaced by a deep neural network working as a universal function approximator. At a high level, the neural network will take camera image as an input and provide the required yaw rate for tracking the path in the next time step.

## Evaluation Metrics

Apart from the fact that more work needs to be put in increasing the robustness of the network itself, methods of verifying the robustness are also required. These methods, rather than being singular numeric values, should be more intuitive about the internal dynamics and the system's characteristics. Along with specifying how good a data driven controller is, these methods also need to point towards specific areas for improvement or regions where the controller falls short of performance. In the following work, two metrics for gauging the cornering performance of a skid-steered robot performing path tracking are developed. The first metric shares an insight on performance similarities between multiple controllers and the second metric provides an intuition on how well a deep learning controller has generalized from the training data.

**Accuracy Metric: Divergence from Benchmarked Values** Traditionally, the performance of a neural network is computed by evaluating statistical error values like the root mean square error (RMSE) and mean squared error (MAE) [4]. The challenge with using these values for a checking performance of a control system is that they do not provide an intuition about how the system's characteristics like its inertia, slip angles and terra-mechanics are impacting the controller's performance. For instance, the linear velocity and yaw rate values predicted by the neural network are simply a function of the tracking error value and are decoupled from the system dynamics. This motivates the development of the first evaluation metric which compares angular acceleration of a vehicle at different curvature values. In contrast to angular

velocity, angular acceleration captures the effects of the error as well as the system properties like inertia as shown in equation 3.

$$\ddot{\psi} = f\left(error, system\ properties\right) \tag{3}$$

Different curvatures of the track are then segregated in ten different bins. Associated with each bin is a probability distribution for the expected angular acceleration values. From figure 7(a), it can be observed that the distribution of the angular acceleration increases as the curvature increase. All the samples are then separated in 10 equidistant bins and mean and standard deviation of angular acceleration value for each bin is computed.

A Kullback-Leibler (KL) divergence [3] is computed between the values predicted by the data driven control techniques and the ones benchmarked by the PID controller. For two probability distributions P and Q, the KL divergences computes proximity between P and Q with the following relationship

$$KL\left(P,Q\right) = \log\left(\frac{\sigma_q}{\sigma_p}\right) + \frac{{\sigma_p}^2 + \left(\mu_p - \mu_q\right)^2}{2{\sigma_q}^2} - \frac{1}{2} \tag{4}$$

Where, $\mu$ is the mean of distribution and $\sigma$ is the standard deviation. Subscripts p, q are for distribution P and Q respectively.

**Generalization Metric: Difference in Bending Energy** Overfitting is common problem for training deep neural networks. It is a condition when the neural network learns only the training data and has a problem generalizing it over unseen data [10]. In this visual servoing task it is likely that the data driven controller may learn exact values of yaw rate as per training data and fail to perform a smooth interpolation between extreme data points resulting in a suboptimal function approximation. With this in mind, an evaluation metric computing the total bending energy of the realized path of the vehicle can be computed.

The change in curvature from any instant $\kappa_i$ to next in instance $\kappa_{i+1}$ is proportional to bending energy of the curve. We denote this change by $|\delta\kappa_i|$. Summation of $|\delta\kappa_i|$ over the entire track is proportional to the total bending energy of the path.

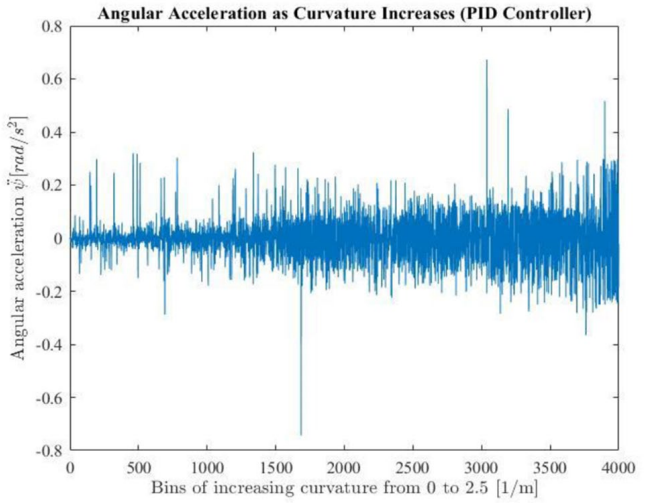
$$\sum\left|\delta\kappa_i\right| \propto Total\ bending\ energy \tag{5}$$

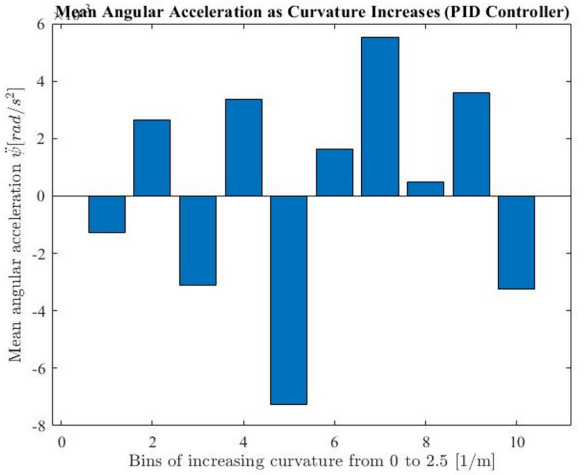
The higher this value, more sharp curves your vehicle undertakes while traversing the track. A lower value is indicative of smoother transition between two consecutive tracking points.

# Data Driven Learning Algorithms

**Imitation Learning** Imitation learning or behavior cloning is a data driven end-to-end supervised learning approach in which the neural network based controller learns

**FIGURE 7** (a), 7(b), 7(c) [top to bottom]. Instantaneous angular acceleration as curvature increase, mean of instantaneous angular acceleration as curvature increases, standard deviation of angular acceleration as curvature increases



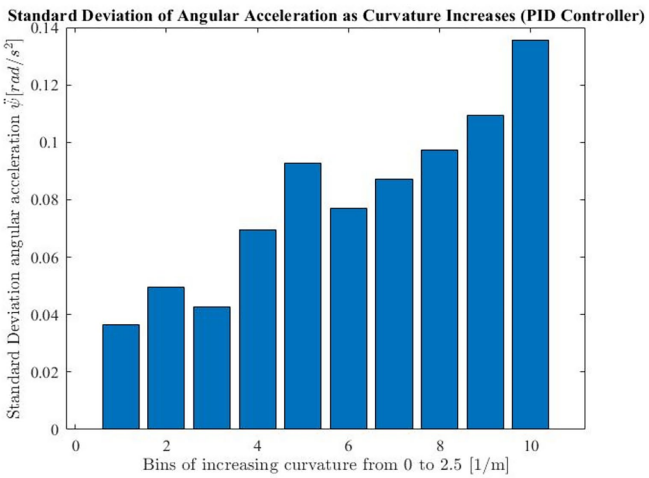
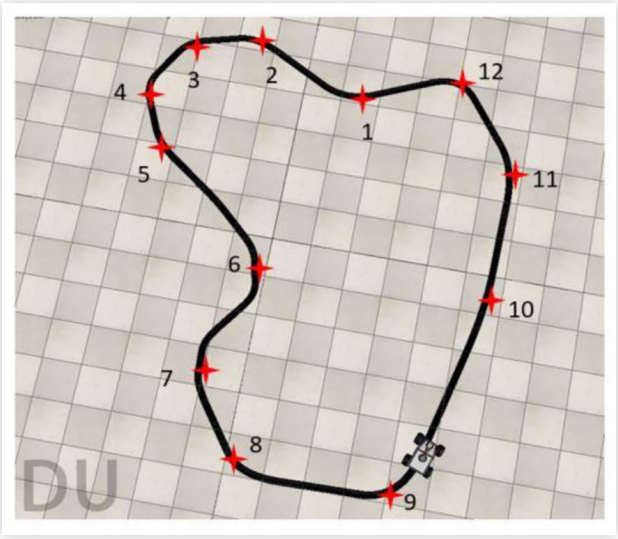



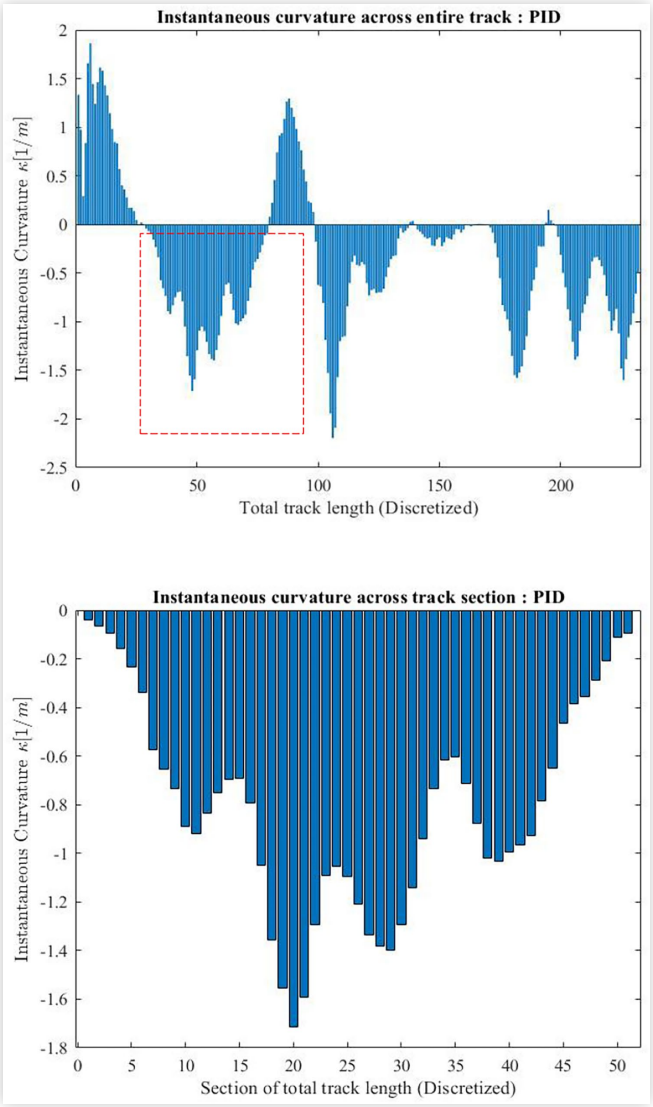
**FIGURE 8** An illustration of the sharp corners of the track. The 12 peaks in figure 9(a) are indicative of the 12 sharp corners illustrated here.




the control actions by associating them with certain learning inputs [2]. In this path tracking example, a neural network is trained to associate each raw image to a desired velocity and yaw rate. The vehicle first goes around the track by collecting structured data of images and associated linear velocity and yaw rate values as provided by the PID controller.

**Training Details** The vehicle went around the track for multiple laps collecting a total of 6000 samples of raw pixel images and the associated velocity values which were to be treated as labels. The training takes place on a 16 layered convolution neural network defined by the hyperparameters in table 1. The input to the neural network is a 32x32 grayscale image and the output is a regression layer predicting linear velocity or yaw rate. The selection of the convolution structure of neural network is inspired from the NVIDIA's neural network tailored for behavior cloning [1]. The choice of optimizer and initial hyperparameters were selected based on M. Abdou et al.'s work [12], which is a sucessions of NVIDIA's work. Hyperparameters were then tuned for improving performance.

**FIGURE 9** (a), 9(b) [top to bottom]. Instantaneous curvature across entire track (positive values for counterclockwise turns and negative for clockwise turns), Zoomed in section of the track enclosed in the red dotted line (height of peaks and valleys are indicatives of sharpness of the turn undertaken by the car)




**Reinforcement Learning** As compared to imitation learning, which is a supervised learning approach, reinforcement learning algorithm learns the control commands by maximizing a user specified reward function [5]. Reinforcement learning leverages the strength of compute power for iterative trial and error approach to guess the correct control values and improves the prediction over time. This is in contrast to supervised learning where the correct control values are provided as identification labels and a loss function is minimized to improve the prediction. A reinforcement learning agent interacts with its environment over numerous iterations and undertakes control actions as a function of some state observations to maximize the user defined reward function over time (figure 11). In an explicit sense, the reinforcement learning agent is trying to learn a function that maps state observations to optimal control actions. In reinforcement learning terms this function is called a policy and denoted by $\pi$.

The training setup has been done similar to the imitation learning and the PID control scheme, with CoppeliaSim being used as a dynamics simulation engine and the training & control algorithm implemented in MATLAB.

**Training Details** Similar to the imitation learning based control, the reinforcement learning algorithm accepts raw pixel images and predicts a yaw rate associated with those images. These input-output features vary in a continuous space (the intensity of any pixel could be anywhere between 0 to 255 and the yaw rate could take any value between the lower and upper limit [-0.825 rad/s ; 0.825 rad/s]) as opposed

**FIGURE 10** Each image is associated with a linear velocity and yaw rate.


**TABLE 1** Training details of imitation learning algorithm

| | |
|---|---|
| Count of training data set images | 16000 |
| Input feature | Image (1024 x 1) |
| Output layer | Regression |
| Network layers | 16 |
| Learning rate | 1e-3 |
| Optimizer | Adam |
| Training : Validation split | 70:30 |
| Total training epochs | 50 |

**FIGURE 11** A schematic diagram for reinforcement learning.

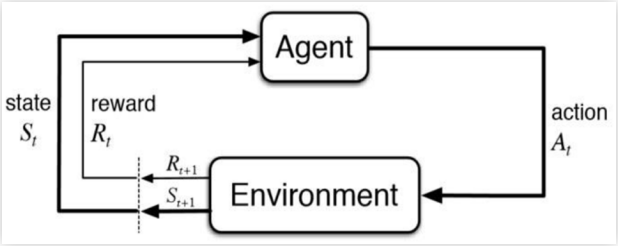



**TABLE 2** Training details of for reinforcement learning algorithm

| | |
|---|---|
| Training agent | Deep deterministic policy agent |
| Observations | [Image; Linear Velocity; Angular Velocity ] |
| Actions | Yaw Rate |
| Simulation environment | CoppeliaSim (synchronous simulation) |
| Target smoothening factor | 1e-3 |
| Discount factor | 0.99 |
| Simulation step size | 10 ms |

to a discrete space where both the input and output features could be from a set of predefined values. This required a reinforcement learning agent that worked with continuous input and output states, leading to the choice of deep deterministic policy (DDPG) agent [6] for training. The DDPG agent was trained with hyperparameters mentioned in table 2.

The reward function in its elementary form needed to be shaped such that the tracking error is minimized.

$$Reward = -error^2 \tag{6}$$

Since the reward computation occurs at each time step, small errors accumulate over time. This leads a trivial optimality condition with equation 6 in which the vehicle does not move forward. To counter this, progression to the next simulation step is provided as a positive feedback and the net reward is calculated as

$$Reward = k * i^2 - m * error^2 \tag{7}$$

Where, i is the number of simulation steps, and k and m are weights for the function parameters.

From a technology specific viewpoint, the reinforcement learning setup was carried out in a synchronous fashion in contrast to the PID and imitation learning control schemes which were done asynchronously. In a synchronous simulation, the simulation engine pauses after every time step till the next control action is received. This was necessary to implement a seamless coordination between CoppeliaSim engine and the MATLAB Reinforcement Learning Toolbox.

# Results

## Divergence

The trained neural network obtained from the imitation learning controller and the policy obtained from the reinforcement learning training were tested for their prediction by simply switching the control scheme block from a PID controller to the trained neural network. The output data was reorganized as required by the evaluation metrics.

The initial observation from the angular acceleration values showed that the imitation learning algorithm behaved similar to the PID controller in the sense that predicted angular acceleration values increased as the track curvature increased. The distribution of the predicted values was slightly offset towards negative values. This can be indicative that the training images were biased with more images with counterclockwise turns. Shuffling the images or adding more images with clockwise turns to the data set could be a few options that can be undertaken to reduce the offset. The second observation is that the imitation learning framework has disregarded the outliers in the data set as peak prediction values are almost half the values of the PID controller. This is indicative that the neural network has generalized over the entire track space rather than learning the exact numeric values.

Comparing the reinforcement learning output to the PID controller, it is evident that the angular acceleration values are different in magnitude. The very high angular acceleration values are indicative that the reinforcement learning controller tries to minimize the error at very high rate which reflects that the weight provided to error term of reward function is very high. Tuning the weights of the reward function, specifically the coefficient of the error term ($m$), will result in reduced angular acceleration values.

Computing the KL divergence of the distributions helps to compare different prediction algorithms by quantifying the proximity in their performance. Figure 13(a) compares how similar the values predicted by the imitation learning

**FIGURE 12** (a), 12(b), 12(c). The instantaneous yaw rates provided by the PID controller, the instantaneous yaw rates predicted by the imitation learning controller, the instantaneous yaw rates predicted by the reinforcement learning controller

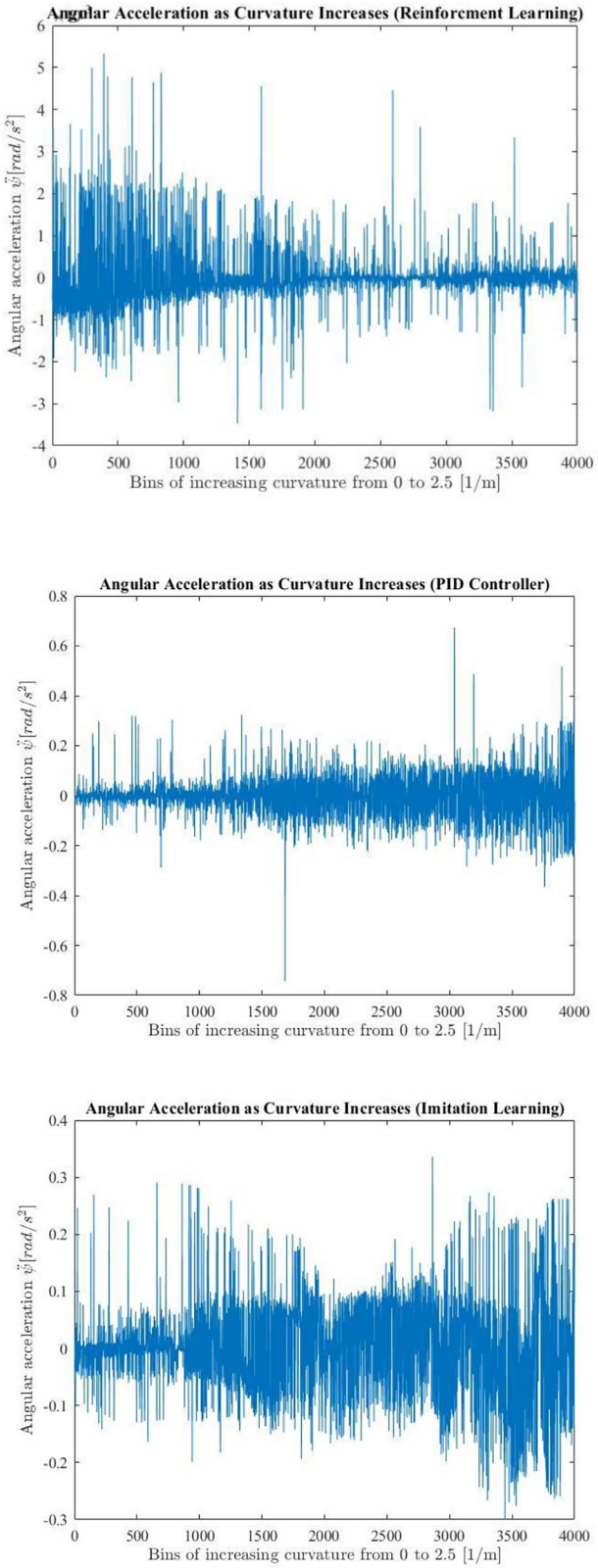
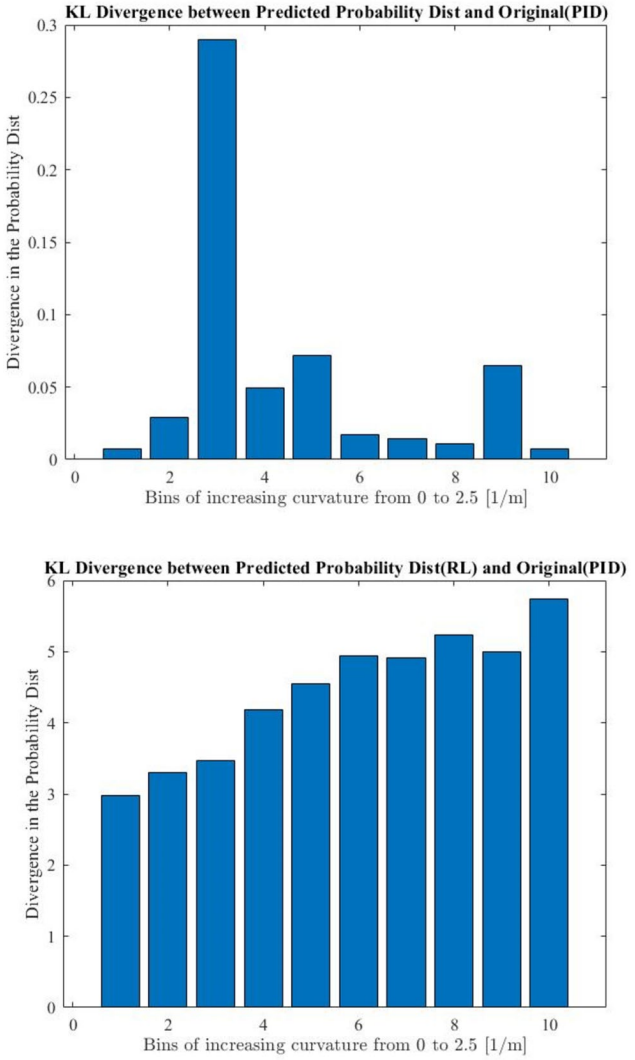



**FIGURE 13** (a), 13(b) [top to bottom]. The divergence of the imitation learning controller from the PID controller, the divergence of reinforcement learning controller from the PID controller.




algorithms are to the values output by the PID controller. Since the imitation learning algorithm is trained using the values provided by the PID controller, it has lower divergence values for different curvature sections. The curvature bin number 3 has the highest divergence for the imitation learning algorithm indicating that the algorithm can now be trained specifically on curvature values from bin number 3 to improve performance.

Since the reinforcement learning agent learns by itself, the predicted values are very different from that of the PID controller and thus diverges more. This by anyway is not indicative of the performance of the algorithm itself as KL

**TABLE 3** Training details of imitation learning algorithm

| | |
|---|---|
| PID Control | 1036 |
| Imitation learning | 970 |
| Reinforcement learning | 13801 |

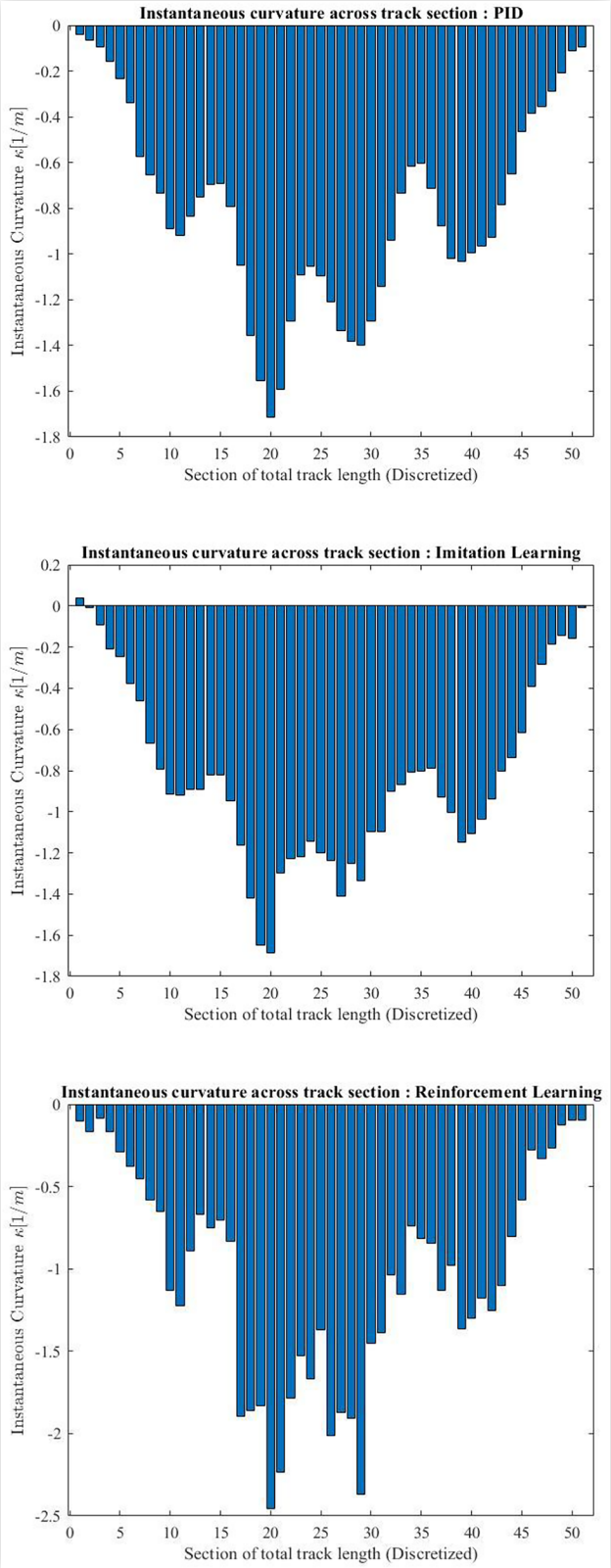



**FIGURE 14** (a), 14(b), 14(c) [top to bottom]. The instantaneous curvature values for three controllers focused only on a certain section of the track.

divergence will only point towards how closely two controllers are performing.

## Total Bending Energy

As mentioned in the Formulation section, the total bending energy is representative of how fast the vehicle changes its turning radius between two time steps. A low bending energy indicates that the vehicles traverses in smooth trajectories where as a high bending energy is result of a zigzag motion. The total bending energy for the three control schemes are shown in table 3.

The values indicate that the imitation learning algorithms generates smoother trajectories even compared to the PID controller on the basis of which it was trained. This is also evident from figure 14 (a) and 14 (b) where imitation learning provides smoother transition from one peak (corner) to other. A possible explanation for this is that since the convolution neural network learns to generalize between set of data points, it tries to fit a smooth curve in the intermediate data points. This is also an indicator that the neural network has done a good job of approximating a function and has not learnt the exact track values (which would indicate overfitting).

For the reinforcement learning algorithm, the high transition in the instantaneous curvature values indicate that the vehicle takes very sharp turns. This can again be a product of poor reward shaping in which high weight to the error term makes it perform like a proportional controller. A strategy similar to PD control scheme can be tried over here where the reward function includes the rate of change of error along with the instantaneous error. This can potentially lower the zigzag motion giving rise to smoother trajectories.

The updated reward function may look like equation

$$Reward = k * i^2 - m * error^2 - n * \left( \dot{error} \right)^2 \tag{8}$$

Where, i is the number of simulation steps, and k, m and n are weights for the function parameters. Tuning these weights will also have an impact on which function parameter dominates the overall performance.

# Conclusions and Future Work

In this work, two evaluation metrics for path tracking algorithms have been discussed. The first metric measures performance proximity of two algorithms and the other is indicative of how the algorithm can regress between extreme tracking points. Along with providing quantitative values of the performance of the algorithm, these metrics also help narrow down specific areas of improvement so that only those could be targeted for performance improvement. Since these metrics are based simply on the input and output values of a

controller, they could be implement for any type of a controller regardless of its internal working.

This work has been established for a mobile robot performing path tracking on a flat surface but given the off-terrain nature of the environment on which skid-steered robots are used, it is imperative to develop these evaluation metrics for those terrains. This future work would involve working with the roll and pitch of the vehicle along with the yaw motion.

# References


1. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B.F. et al., 2016, "End to End Learning for Self-Driving Cars," 1-9. http://arxiv.org/abs/1604.07316.
2. Chella, A., Dindo, H., and Infantino, I., "A Cognitive Framework for Imitation Learning," *Robotics and Autonomous Systems* 54, no. 5 (2006): 403-408. https://doi.org/https://doi.org/10.1016/j.robot.2006.01.008.
3. Contreras-Reyes, J.E. and Arellano-Valle, R.B., "Kullback-Leibler Divergence Measure for Multivariate Skew-Normal Distributions," *Entropy* 14, no. 9 (2012): 1606-1626. https://doi.org/10.3390/e14091606.
4. Japkowicz, N. and Shah, M., *Evaluating Learning Algorithms: A Classification Perspective* (Cambridge University Press, 2011). https://doi.org/10.1017/CBO9780511921803
5. Kober, J., Bagnell, J.A., and Peters, J., "Reinforcement Learning in Robotics: A Survey," *The International Journal of Robotics Research* 32, no. 11 (2013): 1238-1274. https://doi.org/10.1177/0278364913495721.
6. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N.M.O. et al., "Continuous Control with Deep Reinforcement Learning," *CoRR* abs/1509.02971 (2016): n. pag.
7. MATLAB, *Version 7.10.0 (R2010a)* (Natick, Massachusetts: The MathWorks Inc., 2010)
8. Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C. et al., 2015, "Massively Parallel Methods for Deep Reinforcement Learning," *CoRR*, abs/1507.04296, 2015. URL http://arxiv.org/abs/1507.04296.
9. Kumar, P., Spielberg, S., Tulsyan, A., Gopaluni, B. et al., "A Deep Learning Architecture for Predictive Control," *IFAC-PapersOnLine* 51, no. 18 (2018): 512-517. https://doi.org/https://doi.org/10.1016/j.ifacol.2018.09.373.
10. Roelofs, R., Fridovich-Keil, S., Miller, J. et al., "A Meta-Analysis of Overfitting in Machine Learning," *Advances in Neural Information Processing Systems* 32, no. NeurIPS (2019).
11. Rohmer, E., Singh, S.P.N., and Freese, M., 2013, "CoppeliaSim (Formerly V-REP): A Versatile and Scalable Robot Simulation Framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS).*
12. Abdou, M. et al., 2019, "End-to-End Deep Conditional Imitation Learning for Autonomous Driving," in *31st International Conference on Microelectronics (ICM)*, 2019, 346-350, doi:10.1109/ICM48031.2019.9021288.


# Contact Information

**Ameya Salvi**
Graduate Student, Department of Automotive Engineering, Clemson University.
Email: asalvi@clemson.edu

# Acknowledgement

This work was supported by the Automotive Research Center (ARC), a US Army Center of Excellence for modeling and simulation of ground vehicles, under Cooperative Agreement W56HZV-19-2-0001 with the US Army DEVCOM Ground Vehicle Systems Center (GVSC).



ISSN 0148-7191