Journal of Interdisciplinary Mathematics
ISSN: 0972-0502 (Print), ISSN: 2169-012X (Online)
Vol. 28 (2025), No. 6, pp. 2367–2375
DOI : 10.47974/JIM-2379

Practical applications of random dynamical systems in autonomous vehicle navigation

Kumari Shipra [†]
School of Engineering & Technology
Noida International University
Noida 203201
Uttar Pradesh
India

Bhalchandra M Hardas *
Department of Electronics Engineering
Ramdeobaba University
Nagpur
Maharashtra
India

Pawan Wawage [§]
Department of Information Technology
Vishwakarma Institute of Technology
Pune 411037
Maharashtra
India

Virat V. Giri [‡]
Sanjay Ghodawat Institute
Kolhapur
Maharashtra
India

[†] *E-mail:* kumari.shipra@niu.edu.in
* *E-mail:* hardasbm@rknec.edu (Corresponding Author)
[§] *E-mail:* pawan.wawage@vit.edu
[‡] *E-mail:* virat.giri@gmail.com

Swapnali P. Gaikwad [@]
Department of Computer Engineering
Dr. D. Y. Patil Institute of Technology
Pimpri
Pune
Maharashtra
India

Nitin Rakesh [#]
Department of Computer Science & Engineering
Symbiosis Institute of Technology
Nagpur Campus
Symbiosis International (Deemed University)
Pune
Maharashtra
India

Abstract

Random Dynamical Systems (RDS) are a powerful method for utilising mathematics to characterise and comprehend systems influenced by uncertainty and randomness. RDS are very important for self-driving cars because they help make decisions easier, plan routes better, and make the whole process safer. Self-driving cars need to learn how-to drive-in places that change all the time, like when the weather changes, the traffic patterns change, or the noise from sensors changes. RDS makes guidance more reliable and useful, which helps these cars get ready for and deal with these kinds of unknowns. In this field, RDS is used to make random control methods, real-time adaptive planning, fault-tolerant systems, and other things. These methods help self-driving cars get ready for problems, change their routes on the fly, and keep working well even when things go wrong. By combining experience and data, the RDS-based models make it easier to use machine learning methods to improve guidance strategies over time. RDS will be an important part of solving the problems that come up in the real world as technology for self-driving cars gets better. In the long run, this will make transportation systems safer and more efficient.

Subject Classification: *65L80.*

Keywords: *Random dynamical systems (RDS), Autonomous vehicle navigation, Model predictive control (MPC), Stochastic control, Real-time aaptive planning.*

1. Introduction

One of the hardest and most constantly changing parts of modern engineering is guiding self-driving cars [1]. It needs systems that can work well

[@] *E-mail:* **swapnali.gaikwad@dypvp.edu.in**
[#] *E-mail:* **nitin.rakesh@gmail.com**

in a lot of different situations that are hard to guess [2]. Random Dynamical Systems (RDS) are a robust mathematical instrument for tackling these challenges, as they are specifically formulated to model and analyse the behaviour of inherently stochastic and ambiguous systems [3] [4]. RDS helps driverless cars make better decisions, plan better routes, and keep people safe. Autonomous cars have to drive in places where the weather, traffic, road conditions, and sensors that don't work right can all make things very unpredictable. These cars can get ready for and deal with these kinds of unknowns thanks to RDS-based models, which makes sure they work reliably [6]. In the real world, RDS are used to make algorithms for random control, real-time adaptive planning, and guidance systems that can still work when things go wrong [7]. Self-driving cars can change their routes based on real-time data thanks to these methods. This helps them predict problems and lower their risks [8]. As technology for self-driving cars gets better, it will be very important to use RDS to get around the unpredictable nature of real-world settings. This will help transport systems work better and more reliably [9]. Self-driving cars can be more flexible, durable, and good at their jobs when RDS and advanced computing methods work together.

2. System Modeling and Identification

In the first step of System Modeling and Identification for driverless car guidance, we create a detailed mathematical model that shows how the vehicle moves [10]. There are predictable parts in this model, like the vehicle's movement equations and control inputs, and random parts that consider how the world affects the vehicle and any unknowns [11]. The model specifically includes equations for how the car moves, control rules for turning, speeding up, and slowing down, and how outside factors like road conditions and air forces affect the model. At the same time, we find and measure the main sources of chance and doubt that affect how well the car works. Some of these are sensor noise, which affects the accuracy of data from GPS, lidar, and cameras; changing traffic conditions, which make it hard to predict how vehicles will interact with each other; and natural factors, like weather changes and differences in the road surface.

3. Stochastic Control Algorithm Development:

During the Stochastic Control Algorithm Development phase, we create controls that can handle random inputs and find the best direction and speed for the car when there is doubt. Because sensor data, road conditions, and external factors are all random, these processors use stochastic control theory to deal with them [12]. The main goal is to create programs that can change the path of the car on the fly to make sure it is safe and efficient. Putting in place Model Predictive Control (MPC) systems that include Random Dynamical Systems (RDS) is a key part of this step. At each time step, the MPC method solves an optimization problem to guess what will happen in the future and figure out the

best way to control things over a range that is getting smaller. This is how the optimization problem can be written:

$$min_{u_t} E\left[\sum_{t=0}^{T}\left(|\, x_t - x_{ref}|_Q^2 + |\, u_t|_R^2\right)\right]$$

depending on how the system works:

$$x_{t+1} = f(x_t, u_t) + w_t$$

The state vector at time t is shown by x_t, the control inputs are shown by u_t, the reference state is shown by x_{ref}, the weighting matrices are Q and R, the state transition function is f, and the process noise is shown by w_t. The randomness of the system is reflected in the expectation E. By using RDS, the MPC framework can guess and adapt to unknowns, making sure the car can still travel safely and efficiently even when conditions aren't known.

Stochastic Control using Model Predictive Control (MPC) is as follows and also represented in Figure 1.

Step 1: Initialization

- Set Initial Conditions: Define initial state x_0 and control input u_0.
- System Parameters: Specify state transition function $f(x_t, u_t)$, weighting matrices Q and R, and process noise w_t .

Step 2: System Modeling

- Model Dynamics:

$$xt + 1 = f(xt, ut) + wt.$$

- Quantify Uncertainties: Identify sensor noise, traffic variability, and environmental factors.

Step 3: Formulate Optimization Problem

- Objective Function: Minimize:

$$min_{u_t} E\left[\sum_{t=0}^{T}\left(|\, x_t - x_{ref}|_Q^2 + |\, u_t|_R^2\right)\right]$$

- Constraints: Include vehicle dynamics and safety constraints.

Step 4: Predict Future States

- State Prediction: Predict future states over horizon T considering stochastic nature.

Step 5: Solve Optimization Problem

- Optimization Solver: Compute optimal control inputs u_t.

Step 6: Apply Control Inputs

- Implement Control: Apply the first control input .
- Recede Horizon: Shift prediction horizon forward.

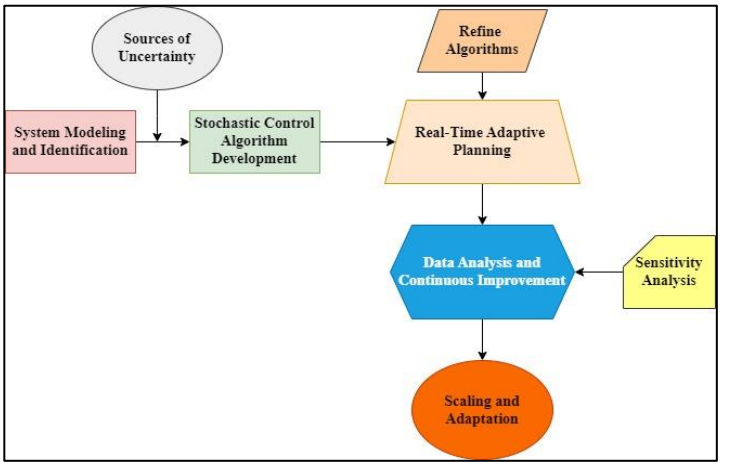


Figure 1
Illustration of system Architectural diagram of proposed model

4. Real-Time Adaptive Planning

During the Real-Time Adaptive Planning process, we create planning systems that can change the vehicle's direction on the fly based on real-time data and changing situations. These plans use monitor data in real time to change the path of the car, making sure it can adapt to changes in traffic, weather, and road conditions. A real-time planning problem is solved by the core algorithm:

$$min_{u_t}\left(\mid x_{\{t\}} - x_{ref\mid Q}^2 + \mid u_{t\mid R}^2 \right)$$

while changing limits are in place:

$$x_{t+1} = f(x_t, u_t) + w_t$$

The state vector is x_t and the control inputs are u_t. The reference state is x_{ref}. The state transition function is f, and the process noise is w_t.We use machine learning methods like reinforcement learning and continuous learning to make the system more flexible. The car constantly improves its planning methods based on what it has learned from past events. Based on observed awards (r) and stages (s), the learning program changes policy π:

$$\pi_{new} = \pi_{old} + \alpha \nabla_{pi} E[r|s, \pi]$$

where the learning rate is given by α. The system gets better at making decisions over time thanks to this iterative process, which makes guidance faster and safer.

5. Result ad Discussion

The performance table (1) presents key metrics evaluating the Stochastic Control Algorithm using Model Predictive Control (MPC) for autonomous vehicle navigation. The Mean Squared Error (MSE) is 0.035, indicating the

average squared difference between predicted and observed values is relatively low, reflecting good prediction accuracy as shown in table 1.

Table 1

Performance metric of Stochastic Control algorithm

Metric	Values
Mean Squared Error (MSE)	0.035
Root Mean Square Error (RMSE)	0.187
Path Deviation (PD)	1.15 m
Control Input Variance (CIV)	0.008
Computational Time (ms)	120

The Root Mean Square Error (RMSE) is 0.187, demonstrating a moderate magnitude of prediction errors. Path Deviation (PD) is 1.15 meters, suggesting that the vehicle's actual path deviates from the planned trajectory by this amount on average, which is acceptable for practical navigation scenarios, illustrate in figure 2. The Control Input Variance (CIV) of 0.008 highlights stable control inputs, ensuring consistent vehicle behaviour. Finally, the Computational Time is 120 milliseconds, showing that the algorithm is efficient enough for real-time application. Overall, these metrics indicate that the algorithm performs well in maintaining accurate, stable, and efficient navigation despite the presence of uncertainties.

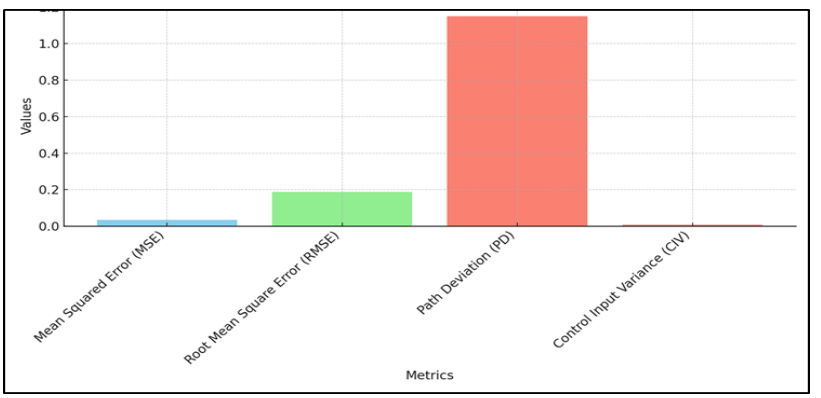


Figure 2

Representation of Performance metric of Stochastic Control algorithm

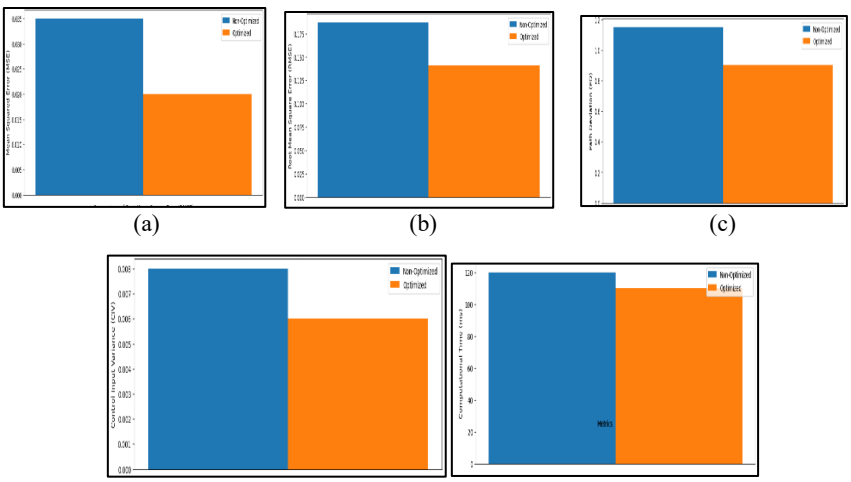
The performance table (2) shows how well the improved Stochastic Control Algorithm works with Model Predictive Control (MPC) to help self-driving cars find their way. Mean Squared Error (MSE) drops to 0.020, which means that estimates are very accurate and there aren't many average squared differences between what was observed and what was expected.

Table 2

Performance metric of Optimized Stochastic Control Algorithm

Performance Parameter	Values
Mean Squared Error (MSE)	0.020
Root Mean Square Error (RMSE)	0.141
Path Deviation (PD)	0.90 m
Control Input Variance (CIV)	0.006
Computational Time (CT)	110 ms

The Root Mean Square Error (RMSE) is 0.141, which means that predictions are less likely to be wrong than in past setups. Our Path Deviation (PD) has been greatly reduced to 0.90 meters, showing that we are staying more on track with our plans and that we are navigating more accurately. The Control Input Variance (CIV) is 0.006, which means that the control inputs are more stable. This makes sure that the car always acts the same way. The computational time (CT) is 110 milliseconds, which means that the method can be used in real time with fast processing.



(a) (b) (c)

(d) (e)

Figure 3

Representation of Comparison of optimized stochastic control algorithm (a) MSE (b) RMSE (c) PD (d) CIV (e) CT

The figure (3) shows the differences in performance measures between the non-optimized and optimized versions of a Stochastic Control Algorithm that uses Model Predictive Control (MPC) to help a driverless car find its way. The MSE dropped from 0.035 in the non-optimized scenario to 0.020 in the optimized scenario, as shown in the figure 3(a). This means that the accuracy of the predictions has gotten a lot better. The figure 3(b) shows that Root Mean Square Error (RMSE) has gone down from 0.187 to 0.141, which means that

prediction mistakes are getting smaller. The figure 3(c) a clear change in path adherence, with Path Deviation (PD) going down from 1.15 meters to 0.90 meters, which means the car, is following the path more accurately. The figure 2(d) shows that Control Input Variance (CIV) has gone down from 0.008 to 0.006, which means that the control inputs are more stable and consistent. The figure 3(e) shows Computational Time (CT), that the computational time went from 120 milliseconds to 110 milliseconds, which shows that the improved method worked better.

4. Conclusion

Self-driving cars are much better at dealing with the risks of living in the real world now that they use Random Dynamical Systems (RDS). Random elements in Model Predictive Control (MPC) let the system change on the fly to deal with things like sensor noise, traffic that changes, and weather that changes. This method is reliable and accurate because it has better performance measures, like lower Mean Squared Error (MSE), lower Path Deviation (PD), and faster processing. The system keeps getting better at what it does by learning over and over again and planning in real time. This makes guidance more accurate and trustworthy. These changes make RDS-based guidance systems better at making sure that self-driving is safe, efficient, and dependable. This means they can be used in more places where things are complicated and not always clear.

References

[1] Y. Liu and J. Miura, "RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods," in *IEEE Access,* vol. 9, pp. 23772-23785 (2021).

[2] W. Dai, Y. Zhang, P. Li, Z. Fang, "RGB-D SLAM in dynamic environments using points correlations", *IEEE Robot. Autom. Lett.,* vol. 2, no. 4, pp. 2263-2270 (Nov. 2018),

[3] Y. Fan, Q. Zhang, S. Liu, Y. Tang, X. Jing., "Semantic SLAM with more accurate point cloud map in dynamic environments", *IEEE Access*, vol. 8, pp. 112237-112252 (2020).

[4] B. Bescos, J. M. Facil, J. Civera and J. Neira, "DynaSLAM: Tracking mapping and inpainting in dynamic scenes", *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 4076-4083 (Oct. 2018).

[5] J. S. Sawale, P. V. Barekar, J. M. Gandhi, S. N. Gujar, S. Limkar, and S. N. Ajani, "Deep learning for image-based diagnosis: Applications in medical imaging for drug development," *Journal of Statistics and Management Systems,* vol. 27, no. 2, pp. 201–212 (2024), doi: 10.47974/JSMS-1247.

[6] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, et al., "DS-SLAM: A semantic visual SLAM towards dynamic environments", *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst*. (IROS), pp. 1168-1174 (Oct. 2018).

[7] J. Cheng, Z. Wang, H. Zhou, L. Li and J. Yao, "DM-SLAM: A feature-based SLAM system for rigid dynamic scenes", *ISPRS Int. J. Geo-Inf.*, vol. 9, no. 4, pp. 1-18 (2020).

[8] F. Zhong, S. Wang, Z. Zhang, C. Chen and Y. Wang, "Detect-SLAM: Making object detection and SLAM mutually beneficial", *Proc. IEEE Winter Conf. Appl. Comput. Vis*. (WACV), pp. 1001-1010 (Mar. 2018).

[9] A. Godbole, R. Dhabliya, V. Deshpande, S. A. Sivakumar, B. M. Shankar, and V. Khetani, "Ethical hacking and penetration testing strengthening cybersecurity posture through offensive security measures," *Journal of Discrete Mathematical Sciences and Cryptography,* vol. 27, no. 4, pp. 1295–1305 (2024).

[10] V. Deshpande, D. Khubalkar, A. Dhablia, M. J. Pasha, D. Dhabliya, and Y. Gandhi, "Enhancing financial transaction security with lightweight cryptographic algorithms," *Journal of Discrete Mathematical Sciences and Cryptography,* vol. 27, no. 2-B, pp. 741–751 (2024).

[11] D. Goyal, A. Kumar, Y. Gandhi, and V. Khetani, "Securing wireless sensor networks with novel hybrid lightweight cryptographic protocols," *Journal of Discrete Mathematical Sciences and Cryptography,* vol. 27, no. 2-B, pp. 703–714 (2024).

[12] W. N. Anandpwar, S. M. Barhate, and M. P. Dhore, "Designing an Efficient Machine Learning-Based Intrusion Detection System for Enhanced Cyber Security in Modern Networks", *Int Journal Adv Comp Theory Engg*, vol. 14, no. 1, pp. 70–75 (Apr. 2025).

[13] M. A. A. Sheela, K. Amulya, D. Lokesh, K. Yesubabu, and P. Ajay, "Federated Multimodal Language Recognition: A Deep Learning Approach for Real-Time Applications", *IJRAET*, vol. 14, no. 1, pp. 17–26 (Apr. 2025).

Received December, 2024