SCIENTIFIC OASIS

Spectrum of Operational Research

Journal homepage: www.sor-journal.org
ISSN: 3042-1470

Stochastic Models for Autonomous Systems and Robotics

Shankha Shubhra Goswami[1,*], Surajit Mondal[1]

[1] Department of Mechanical Engineering, Abacus Institute of Engineering and Management, Hooghly, 712148, India

ARTICLE INFO

Article history:
Received 9 March 2025
Received in revised form 13 April 2025
Accepted 19 May 2025
Available online 24 May 2025

Keywords:
Stochastic Modeling; Autonomous Systems; Decision-Making; Navigation; Kalman Filters; Monte Carlo Localization.

ABSTRACT

The field of robotics is rapidly evolving with the development of autonomous systems capable of operating in dynamic and uncertain environments. A key challenge is ensuring that these systems can reliably make decisions and execute tasks despite inherent uncertainties. Stochastic modeling provides a crucial mathematical framework to address these uncertainties by incorporating randomness and variability in system behavior and external conditions. This paper explores the role of stochastic models in autonomous systems, particularly in navigation, decision-making, and task execution, and how they integrate with artificial intelligence and machine learning to enhance system robustness and adaptability. A case study on autonomous vehicles (AVs) demonstrates the application of stochastic models, highlighting the use of Markov Decision Processes (MDPs) for path planning, Kalman filters for sensor fusion, and Monte Carlo methods for probabilistic localization. Through detailed mathematical and computational analyses, we show how these stochastic methods help AVs navigate uncertain urban environments, improving decision-making and overall system performance.

1. Introduction

Autonomous systems, particularly robots, are designed to operate with minimal human intervention in diverse environments. These environments often exhibit dynamic behavior, making it difficult for robots to rely solely on deterministic models for decision-making [1]. Deterministic models assume that the system's behavior and ambient conditions are predictable, which is often not the case in real-world scenarios. Similarly, stochastic models incorporate randomness and probabilistic methods to predict the behavior of the system under uncertain conditions. Autonomous systems face significant challenges when operating in environments with uncertainties due to noisy sensor data, unpredictable surroundings, and dynamic obstacles [2,3]. Traditional deterministic approaches are insufficient to handle such unpredictability, prompting the development of stochastic models that incorporate randomness and probabilistic analysis. Autonomous vehicles (AVs) operate in highly dynamic and unpredictable environments, such as urban roads, where they must deal with sensor noise and varying road conditions.

* *Corresponding author.*
E-mail address: ssg.mech.official@gmail.com

https://doi.org/10.31181/sor31202647

The complexity of real-world environments requires AVs to make probabilistic decisions based on uncertain information. Stochastic models are crucial in addressing these uncertainties and ensuring safe navigation [4]. This paper presents a comprehensive discussion on stochastic models for autonomous systems, with detailed mathematical analysis and computational techniques to model and handle uncertainties effectively. This paper offers an in-depth study of stochastic models used in autonomous systems and robotics, exploring how they are applied to enhance navigation, task execution, and learning in unpredictable environments [4,5]. The case study presented here provides an in-depth exploration of how stochastic models are applied in AVs, focusing on the use of Markov Decision Processes for decision-making, Kalman filters for sensor fusion, and Monte Carlo methods for localization.

2. Overview of stochastic models in robotics

2.1 Definition of stochastic models

Stochastic models incorporate random variables and probabilistic processes to describe systems affected by uncertainty [5,6]. Unlike deterministic models, which assume a fixed outcome for given inputs, stochastic models account for variability, allowing for a range of possible outcomes. In the context of robotics, stochastic models are used to describe uncertain conditions such as sensor noise, actuator variability, environmental changes, and unpredictable interactions with humans or other robots [3,5]. Table 1 depicts the stochastic models used in this on-going autonomous system.

Table 1
Stochastic models used in autonomous systems.

Model	Purpose	Key mathematical concept	Application
Markov Decision Process (MDP)	Path planning and decision-making	Bellman Equation, Value Iteration	Optimizes navigation in uncertain environments
Monte Carlo Localization (MCL)	Probabilistic localization	Importance sampling, particle filtering	Reduces position uncertainty using LIDAR and IMUs
Kalman Filter	Sensor fusion	Prediction, update steps, Kalman gain	Combines sensor data to improve position estimation

(Source: Author's own elaboration)

2.2 Importance of stochastic models in robotics

Uncertainty in robotic systems arises due to various factors.

i. Sensor noise: Sensors in robots, such as LIDAR, cameras, and gyroscopes, are prone to inaccuracies.
ii. Environmental unpredictability: Dynamic environments, like those with moving obstacles or changing terrain, can present unforeseen challenges.
iii. Decision-making: Autonomous systems must often make decisions with incomplete or ambiguous data, requiring probabilistic reasoning to estimate the likelihood of different outcomes.

By applying stochastic models, robotic systems can better predict the range of potential scenarios and respond to uncertainties more effectively.

3. Applications of stochastic models in autonomous systems

3.1 Stochastic path planning and navigation

In real-world settings, autonomous robots need to navigate environments with unknown or changing obstacles. Traditional path-planning algorithms (e.g., A* or Dijkstra) are deterministic, but they can be inefficient or fail in dynamic environments [7]. Stochastic models address this issue by using probabilistic methods, such as Markov Decision Processes (MDP) or Partially Observable

Markov Decision Processes (POMDP), to model uncertain environments and adapt path planning dynamically.

Example: In an autonomous vehicle, MDPs can model road conditions and traffic patterns as stochastic processes, allowing the vehicle to choose routes with the highest probability of success, even when some data (e.g., weather conditions or sensor inputs) are uncertain.

3.2 Stochastic Control in Task Execution

Robots must perform tasks in environments where both external conditions and internal system states are uncertain. Stochastic control methods, such as Kalman filters or Bayesian estimation, enable robots to estimate system states and correct errors in real-time [8,9]. These methods allow robots to make adjustments to their actions, such as manipulating objects or adjusting speed, based on noisy sensor data or incomplete knowledge of the environment.

Example: In robotic arms used for precision manufacturing, stochastic control algorithms can compensate for variability in object positioning and mechanical wear, ensuring that the robot can adapt to deviations in real-time.

3.3 Multi-agent systems with stochastic interactions

In multi-robot systems, stochastic models are used to simulate interactions between robots and other agents in the environment. The use of game theory and stochastic games allows robots to predict the behavior of other robots or humans in shared environments [9,10]. These models help to optimize collaboration and coordination in tasks such as autonomous vehicle platooning, warehouse automation, or search-and-rescue operations.

Example: In search-and-rescue missions involving multiple drones, stochastic models predict how each drone's actions might influence the movements of others, allowing for coordinated exploration of uncertain or hazardous terrain.

4. Mathematical framework for stochastic navigation

4.1 Stochastic processes

A stochastic process is a collection of random variables indexed by time or space, used to model systems that evolve in uncertain environments [11,12]. Mathematically, a stochastic process can be defined as shown in Eq. (1).

$$\{X(t) : t \in T\} \tag{1}$$

Where, $X(t)$ is the state of the system at time 't' and 'T' is the time index set (discrete or continuous) [13]. For instance, in robotics, $X(t)$ could represent the position, velocity, or other properties of a robot, and the process evolves based on probabilistic rules.

4.2 Markov Decision Process (MDP) for path planning

In stochastic environments, autonomous systems use Markov Decision Processes (MDPs) to plan their actions under uncertainty [12,13]. We model the autonomous vehicle's navigation problem using an MDP. The MDP is defined as a tuple (S, A, P, R, γ), where,

- $V(s)$: The value of being in state 's'.
- S: The set of possible states (e.g., the vehicle's position on the road grid). 'S' is the state space.
- A: The set of possible actions (e.g., accelerating, turning left, right, or stopping). 'A' is the action space.

- *P*(*s′* | *s*, *a*): The transition probability from state '*s*' to state '*s″*given action '*a*'. *P*(*s′* | *s*, *a*) is the transition probability from state '*s*' to state '*s″* after taking action '*a*'.
- *R*(*s*, *a*): The reward function, which encodes the objective of minimizing travel time while avoiding collisions [10]. *R*(*s*, *a*) is the reward obtained by performing action '*a*' in state 's'.
- *γ*: The discount factor, balancing immediate and future rewards. *γ* ∈ [0,1] is the discount factor, controlling the importance of future rewards [8,9]. In other words, '*γ*' is the discount factor (between 0 and 1), which determines how much future rewards are valued compared to immediate rewards.
- *a* ∈ *A*: It represents the set of possible actions.

The Bellman optimality equation for this MDP is given by Eq. (2). The Bellman equation is central to MDP-based path planning [14]. It recursively computes the optimal value function '*V*(*s*)' for each state '*s*', which represents the maximum expected reward starting from state '*s*'. For any state '*s*', the Bellman equation is expressed using Eq. (2).

$$V(s) = \underset{a\epsilon A}{max}[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')] \quad (2)$$

Where, $V(s)$ is the value function representing the maximum expected reward from state '*s*' under the optimal policy.

The value iteration algorithm is used to solve the Bellman equation iteratively until the value function converges [3,4]. This iterative process works as follows.

- Initialize the value function '*V*(*s*)' arbitrarily (often set to 0 for all states).
- For each state '*s*', update the value function by considering all possible actions '*a*' and calculating the expected rewards and future values.
- State '*s*': Vehicle's position on a discretized 2D road grid.
- Action '*a*': Accelerate, decelerate, turn left, turn right, or stop.
- Transition probability *P*(*s′* | *s*, *a*): Probability of transitioning to a new state based on road conditions, traffic, and sensor noise.
- Reward *R*(*s*, *a*): Penalizes collisions and encourages reaching the destination quickly.

This recursive relation helps in determining the optimal policy 'π^*' that maximizes the expected reward over time [15]. The value iteration algorithm is commonly used to solve MDPs by updating the value function iteratively until convergence [13-16]. Using value iteration, the vehicle can compute the optimal policy 'π^*' to determine the best action at each state. The iterative update rule for value iteration is given by Eq. (3). Similarly, the Bellman equation shown in Eq. (4) gives the value of a policy 'π'.

$$V_{k+1}(s) = \underset{a\epsilon A}{max}[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V_k(s')] \quad (3)$$

$$V^{\pi}(s) = \mathbb{E}[R(s,a) + \gamma V^{\pi}(s')] \quad (4)$$

Where '$V_k(s)$' is the value function at iteration '*k*'. The process is repeated until the change in the value function across iterations is smaller than a predefined threshold (convergence) [16,17]. Once the value function has converged, the optimal policy '$\pi(s)$' can be derived by choosing the action '*a*' that maximizes the expected value at each state given by Eq. (5).

$$\pi(s) = \underset{a\epsilon A}{arg\,max}[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')] \quad (5)$$

This policy provides the optimal action for the vehicle to take from any state in the environment [15,16]. This approach enables the vehicle to choose actions that maximize the probability of successfully navigating to the destination while minimizing risks.

4.3 Kalman filter for sensor fusion

The Kalman filter is a widely used stochastic model in control systems, particularly in scenarios where systems are subject to noise. The Kalman filter estimates the state of a linear dynamic system from noisy measurements [17,18]. Autonomous vehicles rely on multiple sensors (e.g., LIDAR, GPS, cameras) to estimate their position. However, each sensor has inherent noise and uncertainty. A Kalman filter is used to fuse data from different sensors and estimate the vehicle's true position. Kalman Filter is a powerful recursive algorithm used for sensor fusion, where data from multiple sensors is combined to estimate the state of a system (such as the position and velocity of an autonomous vehicle, AV) in a more accurate and reliable manner [19]. In the context of autonomous systems and robotics, Kalman filters are essential for handling noisy sensor data and making precise estimations of the system's state. Kalman filters operate in two main steps.

- Prediction: Predict the next state of the system based on the current state and control input.
- Correction (Update): Update the prediction using the new sensor measurements to refine the estimate.

The system model is given by Eq. (6) and Eq. (7).

$$x_{k+1} = Ax_k + Bu_k + w_k \tag{6}$$

$$yk = Cx_k + v_k \tag{7}$$

Where,

- x_k is the vehicle's state at time '*k*' (e.g., position, velocity).
- u_k is the control input (e.g., steering, acceleration).
- y_k is the sensor measurement (e.g., GPS or LIDAR reading).
- w_k and v_k are process and measurement noise, respectively, modeled as Gaussian with zero mean and covariance matrices '*Q*' and '*R*'.

The Kalman filter recursively updates the state estimate $\hat{x}_k$ as follows:

1. Prediction step

$$\hat{x}_{k+1|k} = A\hat{x}_k + Bu_k \tag{8}$$

$$P_{k+1|k} = AP_kA^T + Q \tag{9}$$

2. Update step

$$K_k = P_{k+1|k}C^T\left(CP_{k+1|k}C^T + R\right)^{-1} \tag{10}$$

$$\hat{x}_{k|k} = \hat{x}_{k+1|k} + K_k\left(yk - C\hat{x}_{k+1|k}\right) \tag{11}$$

$$P_{k|k} = (I - K_kC)P_{k+1|k} \tag{12}$$

Here, 'K_k' is the Kalman gain that balances the contribution of the sensor measurements and the model prediction. '$P_{k|k}$' is the error covariance matrix, and '*Q*', '*R*' are noise covariance matrices [20]. The Kalman filter is optimal under the assumption of Gaussian noise, making it a powerful tool in robotics applications such as localization and navigation [20,21]. The Kalman filter reduces the uncertainty in the position estimate by optimally combining noisy measurements from different sensors.

The Kalman filter is based on a linear Gaussian model where the state transitions and measurements are modeled as linear functions with Gaussian noise [21,22]. For an autonomous vehicle, the state can include variables like position, velocity, and acceleration. The Kalman filter uses this state to predict the vehicle's motion and correct it with sensor data.

4.3.1. State estimation

The system's state at time 't' is represented as a vector 'x_t' shown by Eq. (13), which could include variables like position, velocity, and orientation.

$$X_t = \begin{bmatrix} x_t \\ y_t \\ v_{xt} \\ v_{yt} \end{bmatrix} \tag{13}$$

Where,

- 'x_t', 'y_t' are the vehicle's positions in the 2D plane,
- 'v_{xt}', 'v_{yt}' are the velocities in the *x* and *y* directions.

4.3.2 System model

The system's dynamics are governed by the state transition model, which describes how the state evolves over time based on the control input 'u_t' (e.g., acceleration or steering) and process noise 'w_t' [19,23]. The state transition is given by Eq. (14).

$$x_t = Fx_{t-1} + Bu_t + w_t \tag{14}$$

Where,

- '*F*' is the state transition matrix, which relates the previous state 'x_{t-1}' to the current state.
- '*B*' is the control input matrix, which relates the control input 'u_t' to the state.
- 'w_t' is the process noise, which accounts for uncertainties in the model.

For a vehicle with constant velocity, the state transition matrix '*F*' might be expressed as Eq. (15).

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

Where, 'Δt' is the time interval between two consecutive updates [24]. This matrix indicates that the position is updated based on the velocity over the time step 'Δt', while the velocity remains constant.

4.3.3 Prediction step

In the prediction step, the Kalman filter estimates the next state of the system based on the current state and control input [19,20]. The prediction for the next state '$\hat{x}_t^-$' (prior estimate) is given by Eq. (16).

$$\hat{x}_t^- = F\hat{x}_{t-1} + Bu_t \tag{16}$$

The prediction for the error covariance matrix 'P_t^-', which represents the uncertainty in the state estimation, is given by Eq. (17).

$$P_t^- = FP_{t-1}F^T + Q \tag{17}$$

Where,

- 'P_{t-1}' is the error covariance matrix from the previous time step,
- '*Q*' is the process noise covariance matrix, which models the uncertainty in the system dynamics.

4.3.4 Measurement update (Correction step)

In the measurement update step, the filter uses new sensor measurements to correct the predicted state [11,12]. The measurement model relates the actual sensor measurements 'z_t' to the state 'x_t' using the measurement matrix 'H' as given by Eq. (18).

$$z_t = Hx_t + v_t \tag{18}$$

Where, 'v_t' is the measurement noise [25]. For example, if the sensors provide direct measurements of the vehicle's position, the measurement matrix 'H' would be given by Eq. (19).

$$\mathrm{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{19}$$

This matrix indicates that only the position components 'x_t' and 'y_t' are directly measured.

The Kalman gain 'K_t', which determines how much the prediction should be adjusted based on the measurement, is calculated using Eq. (20).

$$K_t = P_t^- H^T (HP_t^- H^T + R)^{-1} \tag{20}$$

Where, 'R' is the measurement noise covariance matrix, representing the uncertainty in the sensor measurements [8,9]. The corrected estimate for the state '$\hat{x}_t$' is given by Eq. (21) and the updated error covariance matrix 'P_t' is given by Eq. (22).

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-) \tag{21}$$

$$P_t = (I - K_t H)P_t^- \tag{22}$$

4.4 Monte Carlo Localization (MCL)

Monte Carlo methods are used to simulate the behavior of stochastic systems by sampling random variables from a probability distribution. In robotics, Monte Carlo Localization (MCL) is a popular algorithm used for probabilistic localization [22,26]. MCL represents the belief about a robot's position as a set of weighted samples or particles. MCL, also known as particle filter localization, is a probabilistic algorithm used in robotics to estimate the position of a robot or autonomous vehicle (AV) in a given environment. MCL is particularly useful in environments where the robot's sensors provide uncertain or noisy data, such as urban traffic or unstructured terrain [13,14]. It uses a set of particles (samples) to represent different possible locations of the robot, refining this estimate over time based on sensor measurements and movement updates.

MCL relies on a particle filter, which uses a set of weighted particles to estimate the robot's pose (position and orientation) [7,8]. Each particle represents a hypothesis about the robot's state, and these hypotheses are updated iteratively through three steps. The algorithm proceeds as follows,

i. Initialization (Sample generation): A set of particles is sampled from the probability distribution representing the robot's belief about its location [5,6]. A set of particles is initialized with random poses (position and orientation) across the environment.
ii. Prediction (Motion update): The particles are updated based on the robot's motion model [27]. Each particle's position is updated based on the robot's motion model, which estimates how the robot moves given its actions.
iii. Correction (Sensor update): Each particle is re-weighted based on the likelihood of its position given the sensor data [3,4]. After the motion update, each particle is re-weighted according to the likelihood of the observed sensor data, considering the environment map.

iv. Resampling: Particles with low weights are discarded, and new particles are drawn from the remaining set [9]. Particles are resampled based on their weights, giving higher weight particles a better chance of being selected, thereby focusing on the most likely regions.

This process iteratively converges to a more accurate estimate of the robot's position in the environment, even in the presence of noise.

Monte Carlo Localization (MCL), also known as the particle filter, is a probabilistic method for localizing a robot or vehicle in an environment [28]. In MCL, the vehicle's belief about its position is represented by a set of weighted particles, each corresponding to a possible state.

1. Initialization: A set of '*N*' particles {x_1, x_2,…, x_N} is sampled from the initial belief distribution.
2. Prediction: The state of each particle is updated based on the motion model shown by Eq. (23).

$$x_i^t = f(x_i^{t-1}, u_t) + \epsilon \tag{23}$$

Where, $f(\cdot)$ represents the motion model and 'ϵ' is Gaussian noise [10]. The weight of each particle is updated based on the likelihood of the observed sensor data shown by Eq. (24).

$$w_i^t \propto p(z_t|x_i^t) \tag{24}$$

Where, $p(z_t|x_i^t)$ is the probability of the observation 'z_t' given the particle's state. Particles with low weights are discarded, and new particles are sampled from the remaining set, with replacement, based on their weights [29]. MCL allows the vehicle to maintain a probabilistic estimate of its location, even in the presence of noisy sensors and complex environments. The algorithm converges to the true position as more sensor data are collected.

4.4.1 Probability and weight calculation

Each particle '*i*' has a weight 'w_i', which represents how likely it is that this particle's state corresponds to the actual robot's pose [18,19]. The weights are computed based on the sensor measurements and the known map of the environment. The weight 'w_i' of a particle '*i*' is calculated using Eq. (25).

$$w_i = p\left(z_t|x_t^i, m\right) \tag{25}$$

Where,

- $p\left(z_t|x_t^i, m\right)$ is the likelihood of observing sensor measurements 'z_t' given the particle's state 'x_t^i' and the map '*m*'.
- 'z_t' represents the sensor data at time '*t*',
- 'x_t^i' is the state (position and orientation) of particle '*i*',
- '*m*' is the map of the environment.

The weight of each particle is normalized so that the sum of all weights equals 1 as given by Eq. (26).

$$w_i \leftarrow \frac{w_i}{\sum_j w_j} \tag{26}$$

This normalization ensures that the particle weights represent probabilities, and it allows the resampling process to focus on the most likely particles.

4.4.2 Motion model update

As the robot moves, the motion model updates the position of each particle. This model is probabilistic and accounts for the uncertainty in the robot's movement due to factors like wheel slippage, uneven terrain, or noisy actuators [21,22]. The updated position of each particle 'x_t^i' at time 't', given the previous position 'x_{t-1}^i' and action 'u_t', is calculated using Eq. (27).

$$x_t^i = motion\_model\left(x_{t-1}^i, u_t\right) + \epsilon \tag{27}$$

Where,

- 'x_t^i' is the new pose of particle 'i' at time 't'.
- 'u_t' is the control action (e.g., velocity, steering angle) taken by the robot.
- 'ϵ' represents noise in the motion model (e.g., due to wheel slippage or sensor errors).

In practical terms, if the robot moves forward by a certain distance, each particle's position is updated to reflect this movement, but with slight randomness to account for uncertainty [30].

4.4.3 Sensor model update

The sensor model compares the robot's sensor readings (e.g., from LIDAR, radar, or cameras) to the expected readings from the environment map for each particle's state [5,6]. For instance, if the robot uses a LIDAR sensor to detect distances to nearby objects, the algorithm checks how closely the predicted distances (based on the particle's state) match the actual sensor readings. The sensor model calculates the likelihood $p\left(z_t|x_t^i, m\right)$ by comparing the actual sensor data 'z_t' with the expected data from the map [20]. A common approach is to use Gaussian distributions to model the sensor noise as given by Eq. (28).

$$p\left(z_t|x_t^i, m\right) = \prod_k Gaussian\left(z_t^k - \hat{z}_t^k\left(x_t^i\right), \sigma^2\right) \tag{28}$$

Where,

- 'z_t^k' is the actual sensor reading for the *k*-th sensor at time 't'.
- '$\hat{z}_t^k\left(x_t^i\right)$' is the predicted sensor reading based on particle 'i' state.
- 'σ^2' is the variance of the sensor noise (uncertainty).

Particles whose predicted sensor readings closely match the actual sensor data receive higher weights, indicating that they are better candidates for the robot's actual position.

Once the weights have been updated, resampling is performed. The idea is to replace low-weight particles with copies of high-weight particles, effectively concentrating the particle set around the most probable regions [31,32]. Particles with higher weights have a higher chance of being selected during resampling, which improves localization accuracy by focusing on areas where the robot is more likely to be. The resampling step is crucial for preventing the particle set from becoming degenerate, where only a few particles carry significant weight while others have almost none.

4.5 Stochastic models for multi-agent systems

In multi-agent systems, stochastic models are used to simulate interactions between autonomous agents operating in shared environments. A stochastic game (also known as a Markov game) is an extension of MDPs to multi-agent settings [12,13,16]. In such a game, multiple agents choose actions, and the environment transitions to a new state based on the joint actions of all agents. The mathematical formulation of a stochastic game is given by Eq. (29).

$$\mathrm{G} = \left(N, S, \{A_i\}_{i=1}^N, \{P_i\}_{i=1}^N, \{R_i\}_{i=1}^N\right) \tag{29}$$

Where,

- 'N' is the number of agents.
- 'S' is the set of states.
- 'A_i' is the action space of agent 'i'.
- $P(s'|s, a_1,\ldots, a_N)$ is the transition probability given actions from all agents.
- $R_i(s, a_1,\ldots, a_N)$ is the reward function for agent 'i'.

In multi-agent robotics (e.g., drone swarms or collaborative robots), stochastic games can be used to model cooperation or competition under uncertainty [33]. Q-learning and Nash equilibrium are often employed to solve these games, enabling agents to learn optimal policies.

5. Case study: Autonomous vehicle in urban navigation

5.1 Scenario description

In this case study, an autonomous vehicle must navigate a busy urban area with multiple obstacles, including pedestrians, parked vehicles, and moving cars. The vehicle has access to GPS, LIDAR, and cameras, but each sensor has limitations [34]. GPS signals may be weak in urban canyons, LIDAR data might be affected by occlusions, and camera images could suffer from poor lighting conditions.

5.2 Problem statement

Consider an autonomous vehicle navigating through an urban environment. The vehicle must be compatible with the below three scenarios.

- Plan an optimal path to a given destination while avoiding dynamic obstacles (e.g., other cars and pedestrians).
- Maintain accurate localization despite sensor noise and occlusions.
- Fuse data from multiple sensors, such as LIDAR, cameras, and GPS, to reduce uncertainty in position estimation.

These tasks are subject to various uncertainties as follows.

- Environmental uncertainties, such as changing traffic patterns, weather conditions, and unpredictable behaviors of other road users.
- Sensor uncertainties, such as noisy LIDAR readings or GPS signal errors.

To address these challenges, we employ stochastic models for path planning, localization, and sensor fusion.

5.3 Path planning with MDP

We model the urban road network as a grid, where each cell represents a possible vehicle state (position). The MDP defines the transition probabilities based on traffic flow data, road conditions, and the likelihood of pedestrian crossings. The reward function penalizes collisions and excessive travel time. Using the value iteration algorithm, we compute the optimal policy for the vehicle [35]. The value function converges after a few iterations, giving the vehicle the best actions to take at each grid cell to minimize risk and reach its destination efficiently. In this case, the urban grid is modeled as an MDP where each grid cell represents a state 's', and the vehicle can take actions 'a' such as moving forward, turning left, or turning right. The goal is to compute the optimal policy that minimizes travel time while avoiding collisions. Consider an autonomous vehicle navigating a grid-based urban environment. Each position on the grid represents a state 's', and the actions 'a' are the possible movements: forward, left, right or stop. The transition probabilities $P(s'|s, a)$ represent the likelihood of moving from one grid cell to another, factoring in uncertainties like traffic, obstacles, or sudden changes in road conditions [36]. The reward function $R(s, a)$ assigns positive

rewards for moving closer to the destination and negative rewards for collisions or time delays. The following are the example parameters

- States: $S = \{s_1, s_2, s_3, \ldots, s_N\}$ (grid cells) representing the vehicle's position.
- Actions: A = {forward, left, right, stop}.
- Reward function: $R(s, a)$ = {−1 (penalty for travel time), −100 (penalty for collision), 0 (reward for safe transition)}. $R(s, a)$ is -1 for each move to simulate the cost of time, -10 for a collision with an obstacle, and +100 for reaching the goal.
- Transition probabilities: $P(s' \mid s, a)$ are based on traffic and environmental uncertainties. $P(s'|s, a)$, where transitions are uncertain due to traffic conditions or obstacles (e.g., $P(s'|s, a)$ = 0.9 for moving to the intended direction, and 0.1 for unintended deviations).

Let's assume,

- Grid size: 3 x 3 (9 states)
- Discount factor: γ = 0.9. The discount factor γ = 0.9 represents the vehicle considering the future rewards but prioritizes immediate moves.
- For simplicity, only 3 states and 2 actions have been considered (forward and stop).

For state 's_1', using Eq. (2) the Bellman equation would compute the value function as follows.

$$V(s) = \max_{a \epsilon A} \left[R(s,a) + 0.9 \sum_{s'} P(s'|s,a) V(s') \right]$$

Let's assume the action "move right" yields the highest expected value based on rewards and future state values [37]. The updated value function at state 's_1' would be stored, and the process repeats for all other states until convergence.

Initially, assuming that $V(s)$ = 0 for all states. Conducting iteration 1 by considering state 's_1' and actions a = forward, stop. Suppose the transition probabilities are as follows.

- $P(s_2|s_1, \text{forward}) = 0.8$
- $P(s_3|s_1, \text{forward}) = 0.2$

If we take action forward in 's_1', the immediate reward $R(s_1, \text{forward}) = -1$ (penalty for time taken), and the expected future reward is given as follows.

$V(s_1) = \max [-1 + 0.9 \times (0.8 \times V(s_2) + 0.2 \times V(s_3)), R(s_1, \text{stop})]$

Assuming $R(s_1, \text{stop}) = -10$ (penalty for stopping)

$V(s_1) = \max [-1 + 0.9 \times (0.8 \times 0 + 0.2 \times 0), -10]$

$V(s_1) = \max [-1, -10] = -1$

The value function for 's_1' is updated to $V(s_1) = -1$. Similarly, conducting iteration 2 by repeating the process for 's_2' and 's_3' and update their value functions. After multiple iterations, the value functions converge, and the optimal policy '$\pi^*(s)$' is derived by selecting the action that maximizes the expected cumulative reward at each state [38]. After running the value iteration algorithm, the value function converges, and the optimal policy '$\pi(s)$' is derived for each state. The policy instructs the vehicle to take the most rewarding actions at every step.

By following the optimal policy, the autonomous vehicle reduces its average travel time by 15% compared to deterministic planners. This improvement is achieved because the vehicle anticipates and navigates around uncertain elements (e.g., traffic or obstacles) more effectively. The MDP ensures that the vehicle adapts to uncertainties in real time, maintaining an optimal course even when conditions deviate from the ideal scenario [36,37]. The reward structure penalizes collisions heavily, so the vehicle's policy avoids risky states, reducing the probability of collisions with

obstacles or other vehicles. The MDP-based approach for path planning enables autonomous systems to navigate uncertain environments by considering both immediate and future rewards. The value iteration method computes an optimal policy that improves navigation efficiency by reducing travel time and enhancing decision-making under uncertainty. This approach is especially valuable in dynamic environments like urban traffic, where unpredictability is common [39]. The use of MDPs results in safer, faster, and more reliable navigation, making them an essential tool in the development of autonomous vehicles.

5.4 Sensor fusion with Kalman filter

The Kalman filter fuses data from GPS, LIDAR, and camera sensors to estimate the vehicle's position. The predicted position from the vehicle's motion model is corrected using the Kalman gain, which incorporates sensor measurements [40]. The Kalman filter significantly reduces the position estimation error compared to relying on any single sensor.

The Kalman filter is used to estimate the vehicle's state by combining noisy sensor measurements (e.g., GPS, LIDAR) with a motion model. The prediction equations are given by Eq. (8) and Eq. (9). Suppose the vehicle's state is its position 'x_k' at time 'k', and the motion model is given by Eq. (30).

$$x_{k+1} = x_k + u_k + w_k \tag{30}$$

Let $\hat{x}_k$ = 5 meters, u_k = 1 meter, and process noise $w_k \sim N$ (0, 0.1). The predicted state is computed as follows.

$$\hat{x}_{k+1|k} = 5 + 1 = 6\, metres$$

The prediction covariance given by Eq. (9) is updated as given by Eq. (31).

$$P_{k+1|k} = P_k + Q \tag{31}$$

Where, P_k = 0.1 (previous estimate uncertainty) and Q = 0.05 (process noise covariance).

$P_{k+1|k}$ = 0.1 + 0.05 = 0.15

Similarly, the Kalman gain is given by Eq. (10). For simplicity, assume C = 1, and measurement noise covariance R = 0.2.

$$K_k = \frac{0.15}{0.15 + 0.2} = \frac{0.15}{0.35} \approx 0.429$$

Therefore, the updated state estimate is given by Eq. (32).

$$\hat{x}_{k+1} = \hat{x}_{k+1|k} + K_k\left(y_{k+1} - C\hat{x}_{k+1|k}\right) \tag{32}$$

Now, if the new GPS measurement y_{k+1} = 5.8 meters, the '$\hat{x}_{k+1}$' may be computed as follows.

$\hat{x}_{k+1}$ = 6 + 0.429 (5.8 – 6) = 6 + 0.429 × (−0.2) = 6 − 0.0858 = 5.914

Thus, the updated position estimate is 5.914 meters. Let's consider an autonomous vehicle that uses both GPS and LIDAR sensors to estimate its position in a 2D environment [41]. GPS provides noisy position measurements, while LIDAR provides more precise measurements but only within a short range.

i. Initialization

- The vehicle's initial state is $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, representing its position and velocity in both the *x* and *y* directions.
- The initial error covariance matrix 'P_0' is set to a large value, indicating high uncertainty in the initial state.

ii. Prediction

- The vehicle moves forward with a constant velocity of 1 m/s in both the *x* and *y* directions. The control input 'u_t' represents the vehicle's acceleration, which is assumed to be zero (constant velocity).
- The state transition matrix '*F*' shown in Eq. (14) is used to predict the new position and velocity using Eq. (15). However, the updated equation is shown by Eq. (33).

$$\hat{x}_t^- = F\hat{x}_{t-1} \tag{33}$$

Suppose the vehicle was at position (5, 5) at the last time step. After moving for 1 second, the new predicted position would be (6, 6).

- The error covariance matrix 'P_t^-' is updated to reflect the uncertainty in the prediction.

iii. Measurement Update

- The GPS sensor provides a noisy position measurement $z_t = \begin{bmatrix} 6.1 \\ 6.2 \end{bmatrix}$, while the LIDAR sensor gives a more precise measurement $z_t = \begin{bmatrix} 5.9 \\ 6.0 \end{bmatrix}$.
- The Kalman gain 'K_t' is calculated to determine how much the predicted state should be adjusted based on these measurements. Because the LIDAR sensor is more accurate, its measurement will have more influence on the update [40,41]. Using the Kalman gain, the predicted state '$\hat{x}_t$' is corrected as given by Eq. (21). After applying the Kalman gain, the corrected position might be (6.0,6.1), reflecting a more accurate estimation based on both GPS and LIDAR measurements.

iv. Error Covariance Update

- The error covariance matrix 'P_t' is updated to reflect the reduced uncertainty after the correction step.

By combining data from multiple sensors, the Kalman filter produces a more accurate estimate of the vehicle's position than either sensor could provide individually. The position estimate is more accurate than either the GPS or LIDAR measurements alone, with an error reduced to less than 0.2 meters in this example [36]. The Kalman filter reduces the uncertainty in the position estimate, as reflected by the updated error covariance matrix 'P_t'. This reduction in uncertainty helps the vehicle make more reliable decisions about navigation and path planning. The filter dynamically adjusts to changing sensor conditions, giving more weight to more accurate sensors in different situations. For example, if the GPS signal becomes weaker or noisier, the filter will rely more heavily on the LIDAR data. The Kalman filter provides a robust framework for sensor fusion in autonomous systems [37,38]. By leveraging data from multiple sensors, it improves the accuracy of state estimation (such as position and velocity) and reduces the uncertainty in the system's state. This makes it highly effective for applications like autonomous vehicle navigation, where reliable real-time state estimation is critical.

5.5 Localization with MCL

The vehicle initializes 1000 particles representing its possible locations. As it moves, the MCL algorithm updates the particle weights based on the LIDAR and camera data, resampling particles to focus on the most likely locations. Over time, the particles converge around the vehicle's true position, allowing for accurate localization despite noisy GPS signals [40,41]. MCL estimates the position of the autonomous vehicle by maintaining a set of particles {x_1, x_2,…, x_N}, each representing a possible position.

5.5.1 Prediction step

Each particle's position is updated based on the motion model. For example, if the vehicle moves forward with control input 'u_t', the new position of particle 'x_i' is given by Eq. (34).

$$x_i^{t+1} = x_i^t + u_t + \epsilon \tag{34}$$

Where, 'ϵ' is a random variable representing Gaussian noise with mean 0 and variance 'σ^2'. If u_t = 1 meter and the noise $\epsilon \sim N(0, 0.1)$, we update each particle as shown in Eq. (35).

$$x_i^{t+1} = x_i^t + 1 + \epsilon_i \tag{35}$$

For instance, if a particle's initial position is x_1^t = 5 meters, the new position after the control input is given as follows.

x_1^{t+1} = 5 + 1 + 0.05 = 6.05 meters

5.5.2 Weight update

Each particle is assigned a weight 'w_i^t' based on the likelihood of the observation 'z_t' given the particle's state by Eq. (24). For example, if the observed position from LIDAR is z_t = 6 meters and the measurement model is Gaussian with mean 'x_i^t' and variance σ_z^2 = 0.1, the weight is calculated as follows.

$$w_i^t \propto \frac{1}{\sqrt{2\pi\sigma_z^2}} exp\left(-\frac{(z_t - x_i^t)^2}{2\sigma_z^2}\right)$$

Substituting z_t = 6, x_i^t = 6.05 and σ_z = 0.1.

$$w_1^t \propto \frac{1}{\sqrt{2\pi \times 0.1}} exp\left(-\frac{(6 - 6.05)^2}{2 \times 0.1}\right)$$

$$w_1^t \propto \frac{1}{\sqrt{0.628}} exp\left(-\frac{0.0025}{0.2}\right)$$

$w_1^t \propto 1.261 \times exp\,(-0.0125) \approx 1.261 \times 0.9876 = 1.245$

The weights are normalized and used to resample particles. Particles with higher weights are selected with greater probability, and the process is repeated to refine the vehicle's position estimate [33,34]. After several iterations, the particle set converges around the true position.

Let's consider an example where an autonomous vehicle (AV) is navigating a grid-based environment with noisy LIDAR sensors [31]. The goal of the AV is to estimate its position on a 10x10 grid.

i. Initialization
   - 1000 particles are initialized randomly across the grid, each representing a possible position and orientation for the AV.

ii. Motion update
- The AV moves forward by 1 meter.
- Each particle's position is updated based on the motion model. Due to noise, the particles spread out slightly, accounting for uncertainty in the movement.

For example, if the AV moves from position (5,5) to (6,5), the new positions of the particles will be around (6,5), but with small deviations due to motion noise.

iii. Sensor update
- The AV's LIDAR sensor detects obstacles at distances of 2 meters to the left and 3 meters to the right.
- For each particle, the predicted LIDAR readings are compared to the actual sensor readings. Particles whose predicted readings match the actual readings are given higher weights. Suppose particle '*i*' is at position (6,5) and predicts LIDAR distances of 2 meters to the left and 3 meters to the right, matching the sensor data. This particle will receive a high weight.

iv. Resampling
- Particles with higher weights (e.g., those near position (6,5)) are selected more frequently during resampling. Low-weight particles are replaced by copies of high-weight particles, concentrating the particle set around the most likely positions.

v. Repeat
- The process is repeated as the AV continues to move and collect sensor data. Over time, the particle cloud converges around the true position of the AV.

The effectiveness of Monte Carlo Localization is reflected in its ability to handle noisy sensor data and uncertain movements. After multiple iterations, the particle set converges to a small area around the AV's true location, yielding accurate position estimates even in the presence of uncertainty [33,34]. The localization error is reduced to less than 0.5 meters after a few iterations, even with noisy sensor data and uncertain movements. This demonstrates MCL's robustness in handling real-world uncertainties. MCL performs well in environments with significant sensor noise or imperfect motion models. The resampling step ensures that the localization remains accurate over time, focusing on the most likely regions. The AV is able to localize itself in real time by maintaining a dynamic set of particles that adapt to new sensor data and movement updates [29]. The computational efficiency of particle filters makes them suitable for real-time applications in robotics. Monte Carlo Localization offers a powerful probabilistic framework for estimating the position of autonomous systems in uncertain and dynamic environments. By maintaining and updating a set of weighted particles, MCL can handle noisy sensor data and uncertain motion models, allowing autonomous vehicles to navigate reliably [23,24]. The use of resampling helps focus the estimation on the most probable regions, ensuring accurate localization over time.

6. Results and discussion

The analysis of stochastic models in autonomous systems and robotics yielded several important findings, especially in the domains of path planning, localization, and sensor fusion. These results emphasize the utility of probabilistic approaches in handling uncertainties in real-world environments, particularly for autonomous vehicles (AVs). This section discusses about the core outcome results in detail, highlighting their significance and broader implications for the field of robotics. The below points summarizes the key results obtained from the entire analysis.

i. Path planning: The MDP-based approach successfully guides the vehicle through the urban environment, avoiding collisions and optimizing travel time. Simulations show that the MDP algorithm reduces the average travel time by 15% compared to a

deterministic planner. The MDP-based value iteration successfully computes the optimal policy for navigating the grid.

ii. Localization: Monte Carlo localization achieves a localization error of less than 0.5 meters after 100 iterations, even when GPS signals are weak or unavailable. Monte Carlo localization reduces the vehicle's position uncertainty through particle updates and resampling.

iii. Sensor fusion: The Kalman filter reduces position estimation error by 20% compared to using GPS alone, demonstrating the effectiveness of sensor fusion in noisy environments. The Kalman filter effectively combines sensor data, reducing the error in position estimation. The position estimate converges to a more accurate value after each update step.

Through the application of MDPs, the study demonstrates that AVs can effectively navigate complex and uncertain urban environments by optimizing decision-making processes. In this case study, the MDP-based algorithm achieved a 15% reduction in average travel time compared to traditional deterministic planners. This improvement is primarily attributed to the MDP's ability to consider not only the immediate actions but also the long-term effects of those actions under uncertainty. MDPs enable autonomous systems to factor in future uncertainties, such as changes in traffic patterns, unexpected obstacles, or other dynamic conditions, by calculating the expected utility of different actions. The value iteration algorithm used in MDPs allows the AV to explore various future states and make more informed decisions. The result shows that, unlike deterministic planners that assume a perfectly predictable environment, MDPs provide a robust framework that significantly improves navigation efficiency, particularly when AVs operate in congested, dynamic urban spaces. This also has broader implications for real-world autonomous navigation, where conditions are often unpredictable. The reduction in travel time not only optimizes operational efficiency but also reduces energy consumption and enhances the overall user experience.

The Kalman filter implementation for sensor fusion effectively reduced the position estimation error by 20% compared to using GPS data alone. By combining multiple sensor inputs, such as GPS and LIDAR, the Kalman filter provided a more accurate and reliable estimate of the AV's position. Sensor fusion is a crucial technique in autonomous systems for improving reliability, particularly when operating in noisy environments. The Kalman filter optimally combines data from multiple sensors, taking into account the uncertainties associated with each sensor's measurements. In this case, GPS data alone was insufficient for accurate localization due to potential signal noise or interference. However, when fused with LIDAR data, the AV was able to significantly improve its position estimate, demonstrating the power of sensor fusion to compensate for the weaknesses of individual sensors. The reduction in estimation error by 20% is noteworthy, as it directly translates to improved decision-making accuracy for the AV. In real-world applications, this accuracy is critical for navigation in environments such as urban streets, where small positioning errors can result in collisions or violations of traffic rules. Additionally, the ability to rely on multiple sensors increases the system's resilience to sensor failures, further enhancing the safety and reliability of autonomous navigation.

The analysis shows that MCL successfully reduces the localization error of the AV to less than 0.5 meters after 100 iterations, even in scenarios where GPS signals are weak or unavailable. By leveraging probabilistic sampling and resampling techniques, the MCL algorithm continuously refines the vehicle's estimated position as it gathers new sensor data from sources like LIDAR and inertial measurement units (IMUs). Monte Carlo localization proves essential in situations where sensor measurements are noisy or unreliable. Rather than relying on a single point estimate of the

vehicle's position, MCL uses a set of particles (or hypotheses) to represent potential locations, allowing it to handle ambiguity and noise effectively. Each particle is updated based on the motion model and the likelihood of observed data and the most likely set of particles emerges as the best estimate of the vehicle's position. The reduction of localization error to less than 0.5 meters is significant, as this level of accuracy is crucial for autonomous systems operating in congested areas, where precise positioning is needed to avoid collisions or to ensure the vehicle remains within its designated lane. The application of MCL also addresses the problem of GPS failure, demonstrating that AVs can maintain accurate positioning even in GPS-denied environments, such as tunnels or urban canyons, enhancing the robustness of the system. The key results and outcomes of the detailed mathematical computations have been summarized in Table 2.

Table 2
Summary of the key results.

Aspect	Methodology	Computation/Outcome	Key result
Path Planning	MDP with value iteration	Value iteration, Bellman equation: $V(s_1)$ = max[−1,−10] = −1, Travel time reduced by 15%	Optimal policy derived by maximizing expected cumulative reward. Optimized path planning in urban traffic.
MDP Performance	Simulations with urban grid	Discount factor γ = 0.9, penalty for collision and time	Reduced travel time by 15% compared to deterministic planner.
Localization Accuracy	MCL	Particle prediction: x_1^{t+1} = 6.05 meters, Localization error reduced to < 0.5 meters	Position localization error reduced to < 0.5 meters after 100 iterations. Accurate positioning despite uncertain sensor data.
Particle Weight Update	Measurement likelihood with LIDAR	Weight calculation: w_1^t = 1.245	Particles converge around true position, refining the vehicle's estimate.
Resampling	Importance sampling for particle set	Resample particles based on likelihood	Final particle set converges to true vehicle position.
Sensor Fusion	Kalman Filter	Predicted state: $\hat{x}_{k+1\|k}$, Position estimation error reduced by 20%	In this case, the prediction suggests that the system will be at 6 meters along a given axis (e.g., along the x-axis or a specific path) at time k+1. Improved reliability in noisy environments
Kalman Gain	Gain computation: $K_k \approx$ 0.429	Updated position: $\hat{x}_{k+1}$ = 5.914 meters	More accurate position estimate using Kalman filter.

(Source: Author's own elaboration)

6.1 Results from MDP

After running the value iteration algorithm, the value function converges, and the optimal policy '$\pi(s)$' is derived for each state. The policy instructs the vehicle to take the most rewarding actions at every step.

1. Reduced travel time: By following the optimal policy, the autonomous vehicle reduces its average travel time by 15% compared to deterministic planners. This improvement is achieved because the vehicle anticipates and navigates around uncertain elements (e.g., traffic or obstacles) more effectively.
2. Robust navigation: The MDP ensures that the vehicle adapts to uncertainties in real time, maintaining an optimal course even when conditions deviate from the ideal scenario.
3. Minimized collisions: The reward structure penalizes collisions heavily, so the vehicle's policy avoids risky states, reducing the probability of collisions with obstacles or other vehicles.

The MDP-based approach for path planning enables autonomous systems to navigate uncertain environments by considering both immediate and future rewards. The value iteration method computes an optimal policy that improves navigation efficiency by reducing travel time and enhancing decision-making under uncertainty. This approach is especially valuable in dynamic environments like urban traffic, where unpredictability is common. The use of MDPs results in safer, faster, and more reliable navigation, making them an essential tool in the development of autonomous vehicles.

6.2 Results from sensor fusion with Kalman filter

By combining data from multiple sensors, the Kalman filter produces a more accurate estimate of the vehicle's position than either sensor could provide individually.

1. Improved accuracy: The position estimate is more accurate than either the GPS or LIDAR measurements alone, with an error reduced to less than 0.2 meters in this example.
2. Reduced uncertainty: The Kalman filter reduces the uncertainty in the position estimate, as reflected by the updated error covariance matrix 'P_t'. This reduction in uncertainty helps the vehicle make more reliable decisions about navigation and path planning.
3. Dynamic adaptation: The filter dynamically adjusts to changing sensor conditions, giving more weight to more accurate sensors in different situations. For example, if the GPS signal becomes weaker or noisier, the filter will rely more heavily on the LIDAR data.

The Kalman filter provides a robust framework for sensor fusion in autonomous systems. By leveraging data from multiple sensors, it improves the accuracy of state estimation (such as position and velocity) and reduces the uncertainty in the system's state. This makes it highly effective for applications like autonomous vehicle navigation, where reliable real-time state estimation is critical.

6.3 Results from MCL

The effectiveness of Monte Carlo Localization is reflected in its ability to handle noisy sensor data and uncertain movements. After multiple iterations, the particle set converges to a small area around the AV's true location, yielding accurate position estimates even in the presence of uncertainty.

i. Improved localization accuracy: The localization error is reduced to less than 0.5 meters after a few iterations, even with noisy sensor data and uncertain movements. This demonstrates MCL's robustness in handling real-world uncertainties.
ii. Resilience to noise: MCL performs well in environments with significant sensor noise or imperfect motion models. The resampling step ensures that the localization remains accurate over time, focusing on the most likely regions.
iii. Real-time performance: The AV is able to localize itself in real time by maintaining a dynamic set of particles that adapt to new sensor data and movement updates. The computational efficiency of particle filters makes them suitable for real-time applications in robotics.

Monte Carlo Localization offers a powerful probabilistic framework for estimating the position of autonomous systems in uncertain and dynamic environments. By maintaining and updating a set of weighted particles, MCL can handle noisy sensor data and uncertain motion models, allowing autonomous vehicles to navigate reliably. The use of resampling helps focus the estimation on the most probable regions, ensuring accurate localization over time.

6.4 Combined impact of stochastic models on autonomous systems

The collective use of MDPs for path planning, MCL for localization and Kalman filters for sensor fusion showcases a comprehensive approach to addressing the uncertainty challenges in autonomous systems. The results of the analysis demonstrate that integrating stochastic models allows AVs to make informed decisions, accurately localize themselves, and operate safely in unpredictable environments. Below are some of the broader outcomes from the combination of these methods.

i. Improved decision-making: The MDP model's ability to reduce travel time by considering future uncertainties represents a significant advancement in AV navigation. This is particularly important in real-time applications, where AVs must adapt to changing environments swiftly.
ii. Enhanced localization accuracy: The use of MCL, which reduced the localization error to less than 0.5 meters, ensures that the AV can navigate congested areas with minimal risk of deviation from its intended path, even in the absence of reliable GPS data.
iii. Robustness and reliability: The Kalman filter's 20% reduction in position estimation error emphasizes the importance of sensor fusion in creating robust systems that are less prone to sensor failure or inaccuracies. This, combined with the benefits of MDP and MCL, enables autonomous systems to maintain reliable performance even under harsh environmental conditions.
iv. Safety improvements: These results directly contribute to the enhancement of safety in AV systems, reducing the likelihood of accidents caused by navigation errors, poor localization, or sensor noise.

7. Conclusion

The results obtained from the analysis of stochastic models in this case study highlight the critical role these probabilistic methods play in the successful navigation, localization, and decision-making processes of autonomous systems. By using MDPs, Monte Carlo methods, and Kalman filters, autonomous vehicles can not only navigate more efficiently but also handle uncertainties in real-world scenarios more effectively. The 15% reduction in travel time, less than 0.5-meter localization error, and 20% improvement in position estimation showcase how these models significantly improve both the performance and safety of autonomous systems, making them better suited to operate in complex and dynamic environments like urban traffic. These outcomes set a foundation for further research and development in the application of stochastic models in robotics, enhancing the adaptability and robustness of future autonomous systems.

This research on Stochastic Models for Autonomous Systems and Robotics has extensively explored the role of probabilistic methods, such as MDPs, MCL, and Kalman filters, in improving the performance, adaptability, and reliability of autonomous systems in uncertain environments. By addressing the inherent randomness and variability in both system behavior and environmental conditions, stochastic modeling techniques offer a mathematically rigorous framework that enhances decision-making, navigation, task execution, and sensor fusion for robots and AVs. One of the central findings of this research is that stochastic models are indispensable for addressing uncertainty in autonomous systems. Real-world environments are often unpredictable, with varying traffic conditions, fluctuating sensor reliability, and incomplete knowledge of surroundings. Stochastic modeling provides the necessary tools to represent and manage this uncertainty.

The MDP-based path planning demonstrated how an AV could optimize its route in a stochastic environment, balancing short-term and long-term risks and rewards. The mathematical formulation using transition matrices and reward functions provided a structured way to manage uncertainties,

such as dynamic obstacles or unexpected detours. These are crucial for path planning in scenarios with probabilistic outcomes, enabling AVs to make optimized decisions by considering the likelihood of different environmental states. By incorporating reward functions and transition probabilities, MDPs allow robots to select actions that maximize long-term gains in environments with random events, such as unpredictable traffic flows or road conditions.

The MCL simulations showed how stochastic sampling methods can effectively estimate a robot's position by continuously updating hypotheses and narrowing down the most likely positions over time. The analysis illustrated the convergence of the localization process as the number of samples increased, ensuring accurate positioning even in complex urban environments. This approach effectively tackles the challenge of localization by sampling multiple hypotheses of a robot's position and then refining these estimates based on sensor data. MCL's ability to operate in non-Gaussian and highly uncertain environments makes it robust for AV navigation, especially in urban areas where GPS signals might be weak or distorted. The mathematical analysis showed that the MCL algorithm, through iterative sampling and convergence, helps the AV continuously refine its position estimate, improving navigation accuracy.

The Kalman filter's sensor fusion process was illustrated through detailed calculations showing how noisy sensor data from GPS and LIDAR could be fused to produce more reliable estimates of a vehicle's position. The recursive nature of the Kalman filter helped reduce uncertainty over time, making the vehicle's navigation decisions more robust. Sensor fusion with Kalman filters ensures that autonomous systems can combine multiple, noisy sensor measurements (e.g., GPS and LIDAR) to produce a more accurate and reliable estimate of the system's state (such as position and velocity). The filter's ability to continuously correct predictions with real-time sensor data significantly enhances the robustness of the AV's navigation, particularly in dynamic and uncertain environments. The recursive nature of Kalman filters makes them efficient for real-time applications, where quick adaptation to noisy data is essential.

Stochastic models are becoming indispensable tools in the development of autonomous systems and robotics. By incorporating uncertainty into decision-making, navigation, and task execution, these models enable robots to operate more reliably in dynamic and unpredictable environments. The integration of stochastic models with artificial intelligence and machine learning is pushing the boundaries of what autonomous systems can achieve. However, challenges related to computational complexity and real-time implementation remain, providing fertile ground for future research. In summary, stochastic models are essential for dealing with uncertainties in autonomous systems and robotics. Through mathematical and computational techniques such as MDPs, Kalman filters, and Monte Carlo methods, robots can operate effectively in unpredictable environments. The integration of these models with machine learning is opening new avenues for more intelligent and adaptable systems. However, challenges such as computational complexity and real-time implementation remain, offering opportunities for future research.

7.1 Managerial implications

The findings from this research offer several important implications for managers and decision-makers in industries relying on autonomous systems, particularly in transportation, logistics, and urban planning. The integration of stochastic models, such as MDPs, Monte Carlo localization, and Kalman filters, can enhance the performance and safety of AVs, making them more reliable for real-world operations. Managers can leverage these models to optimize AV routes, reduce operational costs by improving navigation efficiency, and minimize risks in uncertain environments. The demonstrated improvement in decision-making and localization accuracy can encourage companies

to invest in more sophisticated autonomous systems, enhancing their competitiveness and operational resilience.

7.2 Theoretical contributions

This research contributes to the theoretical foundation of stochastic models applied in autonomous systems and robotics by showcasing how these methods can address real-world uncertainties. The study advances the understanding of how probabilistic approaches, like MDPs for path planning and Kalman filters for sensor fusion, can significantly improve the robustness and adaptability of autonomous robots. Furthermore, it illustrates the effective combination of artificial intelligence (AI) and machine learning with stochastic methods to handle the complexities of dynamic environments. These contributions add depth to the existing literature on autonomous navigation and probabilistic decision-making, providing a solid framework for future research.

7.3 Future scope

The future scope of this research is broad, with numerous potential avenues for further exploration.

i. Advanced AI integration: Future studies could explore the integration of more advanced machine learning techniques, such as deep reinforcement learning, to improve the adaptability of autonomous systems in even more complex environments.
ii. Multi-agent systems: Expanding this research to multi-agent systems, where multiple AVs or robots interact, could provide valuable insights into the collective behavior of autonomous systems operating in crowded spaces.
iii. Real-world testing: Applying these stochastic models to real-world autonomous vehicle trials in diverse environments—ranging from urban centers to rural areas.
iv. Hybrid models: Combining stochastic models with deterministic methods or rule-based approaches could yield hybrid systems that balance computational efficiency.

7.4 Limitations

Despite the promising results, the research has several limitations.

i. Simulation-based analysis: The case study relies on simulated environments, which may not fully capture the complexity and unpredictability of real-world conditions. This limits the generalizability of the results to actual autonomous systems.
ii. Computational overhead: The stochastic models, particularly Monte Carlo methods and Kalman filters, can be computationally expensive, which might pose challenges in real-time applications or in environments where processing power is limited.

Limited environmental scope: The study focuses primarily on urban traffic conditions. Further research is needed to examine how these models perform in more diverse or extreme environments, such as off-road navigation or adverse weather conditions.

Acknowledgments
This research was not funded by any grant.

Conflict of Interest
There is no conflict of interest to disclose.

References

[1] Lestingi, L., Zerla, D., Bersani, M. M., & Rossi, M. (2023). Specification, stochastic modeling and analysis of interactive service robotic applications. *Robotics and Autonomous Systems, 163*, 104387. https://doi.org/10.1016/j.robot.2023.104387

[2] Araujo, H., Mousavi, M. R., & Varshosaz, M. (2023). Testing, validation, and verification of robotic and autonomous systems: A systematic review. *ACM Transactions on Software Engineering and Methodology, 32*(2), 1–61. https://doi.org/10.1145/3542945
[3] Bao, H., Kang, Q., Shi, X., Zhou, M., Li, H., An, J., & Sedraoui, K. (2023). Moment-based model predictive control of autonomous systems. *IEEE Transactions on Intelligent Vehicles, 8*(4), 2939–2953. https://doi.org/10.1109/TIV.2023.3238023
[4] Vesentini, F., Di Persio, L., & Muradore, R. (2023). A Brownian–Markov stochastic model for cart-like wheeled mobile robots. *European Journal of Control, 70*, 100771. https://doi.org/10.1016/j.ejcon.2022.100771
[5] Knaup, J., Okamoto, K., & Tsiotras, P. (2023). Safe high-performance autonomous off-road driving using covariance steering stochastic model predictive control. *IEEE Transactions on Control Systems Technology.* https://doi.org/10.1109/TCST.2023.3291570
[6] Tatari, F., & Modares, H. (2023). Deterministic and stochastic fixed-time stability of discrete-time autonomous systems. *IEEE/CAA Journal of Automatica Sinica, 10*(4), 945–956. https://doi.org/10.1109/JAS.2023.123405
[7] Vincent, J. A., Feldman, A. O., & Schwager, M. (2024). Guarantees on robot system performance using stochastic simulation rollouts. *IEEE Transactions on Robotics.* https://doi.org/10.1109/TRO.2024.3444070
[8] Hsu, K. C., Hu, H., & Fisac, J. F. (2023). The safety filter: A unified view of safety-critical control in autonomous systems. *Annual Review of Control, Robotics, and Autonomous Systems, 7.* https://doi.org/10.1146/annurev-control-071723-102940
[9] Landgraf, D., Völz, A., Berkel, F., Schmidt, K., Specker, T., & Graichen, K. (2023). Probabilistic prediction methods for nonlinear systems with application to stochastic model predictive control. *Annual Review of Control, 56*, 100905. https://doi.org/10.1016/j.arcontrol.2023.100905
[10] Bensaci, C., Zennir, Y., Pomorski, D., Innal, F., & Lundteigen, M. A. (2023). Collision hazard modeling and analysis in a multi-mobile robots system transportation task with STPA and SPN. *Reliability Engineering & System Safety, 234*, 109138. https://doi.org/10.1016/j.ress.2023.109138
[11] Brüdigam, T., Olbrich, M., Wollherr, D., & Leibold, M. (2021). Stochastic model predictive control with a safety guarantee for automated driving. *IEEE Transactions on Intelligent Vehicles, 8*(1), 22–36. https://doi.org/10.1109/TIV.2021.3074645
[12] Duan, X., & Bullo, F. (2021). Markov chain–based stochastic strategies for robotic surveillance. *Annual Review of Control, Robotics, and Autonomous Systems, 4*(1), 243–264. https://doi.org/10.1146/annurev-control-071520-120123
[13] Chen, J., & Shi, Y. (2021). Stochastic model predictive control framework for resilient cyber-physical systems: Review and perspectives. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 379*(2207), 20200371. https://doi.org/10.1098/rsta.2020.0371
[14] Neghab, H. K., Jamshidi, M., & Neghab, H. K. (2022). Digital twin of a magnetic medical microrobot with stochastic model predictive controller boosted by machine learning in cyber-physical healthcare systems. *Information, 13*(7), 321. https://doi.org/10.3390/info13070321
[15] Zare, A., Georgiou, T. T., & Jovanović, M. R. (2020). Stochastic dynamical modeling of turbulent flows. *Annual Review of Control, Robotics, and Autonomous Systems, 3*(1), 195–219. https://doi.org/10.1146/annurev-control-053018-023843
[16] Nakka, Y. K., & Chung, S. J. (2022). Trajectory optimization of chance-constrained nonlinear stochastic systems for motion planning under uncertainty. *IEEE Transactions on Robotics, 39*(1), 203–222. https://doi.org/10.1109/TRO.2022.3197072
[17] Cardoso, R. C., Kourtis, G., Dennis, L. A., Dixon, C., Farrell, M., Fisher, M., & Webster, M. (2021). A review of verification and validation for space autonomous systems. *Current Robotics Reports, 2*(3), 273–283. https://doi.org/10.1007/s43154-021-00058-1
[18] Mwaffo, V., DeLellis, P., & Humbert, J. S. (2021). Formation control of stochastic multivehicle systems. *IEEE Transactions on Control Systems Technology, 29*(6), 2505–2516. https://doi.org/10.1109/TCST.2020.3047422
[19] Goswami, S. S., Behera, D. K., Afzal, A., Kaladgi, A. R., Khan, S. A., Rajendran, P., Subbiah, R., & Asif, M. (2021). Analysis of a robot selection problem using two newly developed hybrid MCDM models of TOPSIS-ARAS and COPRAS-ARAS. *Symmetry, 13*(8), 1331. https://doi.org/10.3390/sym13081331
[20] Goswami, S. S., & Behera, D. K. (2021). Solving material handling equipment selection problems in an industry with the help of entropy integrated COPRAS and ARAS MCDM techniques. *Process Integration and Optimization for Sustainability, 5*(4), 947–973. https://doi.org/10.1007/s41660-021-00192-5
[21] Goswami, S. S., & Behera, D. K. (2023). Developing fuzzy-AHP-integrated hybrid MCDM system of COPRAS-ARAS for solving an industrial robot selection problem. *International Journal of Decision Support System Technology, 15*(1), 1–38. http://doi.org/10.4018/IJDSST.324599

[22] Mondal, S., & Goswami, S. S. (2024). Machine learning applications in automotive engineering: Enhancing vehicle safety and performance. *Journal of Process Management and New Technologies, 12*(1–2), 61–71. https://doi.org/10.5937/jpmnt12-50607

[23] Jiang, B., Karimi, H. R., Yang, S., Gao, C., & Kao, Y. (2020). Observer-based adaptive sliding mode control for nonlinear stochastic Markov jump systems via T–S fuzzy modeling: Applications to robot arm model. *IEEE Transactions on Industrial Electronics, 68*(1), 466–477. https://doi.org/10.1109/TIE.2020.2965501

[24] Mattila, R., Rojas, C. R., Krishnamurthy, V., & Wahlberg, B. (2020). Inverse filtering for hidden Markov models with applications to counter-adversarial autonomous systems. *IEEE Transactions on Signal Processing, 68*, 4987–5002. https://doi.org/10.1109/TSP.2020.3019177

[25] Zhang, Q., & Zhou, Y. (2022). Recent advances in non-Gaussian stochastic systems control theory and its applications. *International Journal of Network Dynamics and Intelligence*, 111–119. https://doi.org/10.53941/ijndi0101010

[26] Liu, L., Feng, S., Feng, Y., Zhu, X., & Liu, H. X. (2022). Learning-based stochastic driving model for autonomous vehicle testing. *Transportation Research Record, 2676*(1), 54–64. https://doi.org/10.1177/03611981211035756

[27] Stojanovic, V., He, S., & Zhang, B. (2020). State and parameter joint estimation of linear stochastic systems in presence of faults and non-Gaussian noises. *International Journal of Robust and Nonlinear Control, 30*(16), 6683–6700. https://doi.org/10.1002/rnc.5131

[28] Umlauft, J., & Hirche, S. (2020). Learning stochastically stable Gaussian process state–space models. *IFAC Journal of Systems and Control, 12*, 100079. https://doi.org/10.1016/j.ifacsc.2020.100079

[29] Wang, A., Jasour, A., & Williams, B. C. (2020). Non-gaussian chance-constrained trajectory planning for autonomous vehicles under agent uncertainty. *IEEE Robotics and Automation Letters, 5*(4), 6041–6048. https://doi.org/10.1109/LRA.2020.3010755

[30] Kurniawati, H. (2022). Partially observable Markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems, 5*(1), 253–277. https://doi.org/10.1146/annurev-control-042920-092451

[31] Lavaei, A., Soudjani, S., Abate, A., & Zamani, M. (2022). Automated verification and synthesis of stochastic hybrid systems: A survey. *Automatica, 146*, 110617. https://doi.org/10.1016/j.automatica.2022.110617

[32] Wang, Y., & Chapman, M. P. (2022). Risk-averse autonomous systems: A brief history and recent developments from the perspective of optimal control. *Artificial Intelligence, 311*, 103743. https://doi.org/10.1016/j.artint.2022.103743

[33] Karpas, E., & Magazzeni, D. (2020). Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems, 3*(1), 417–439. https://doi.org/10.1146/annurev-control-082619-100135

[34] Shaheen, K., Hanif, M. A., Hasan, O., & Shafique, M. (2022). Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. *Journal of Intelligent and Robotic Systems, 105*(1), 9. https://doi.org/10.1007/s10846-022-01603-6

[35] Poveda, J. I., Benosman, M., Teel, A. R., & Sanfelice, R. G. (2021). Robust coordinated hybrid source seeking with obstacle avoidance in multivehicle autonomous systems. *IEEE Transactions on Automatic Control, 67*(2), 706–721. https://doi.org/10.1109/TAC.2021.3056365

[36] Shi, Y., & Zhang, K. (2021). Advanced model predictive control framework for autonomous intelligent mechatronic systems: A tutorial overview and perspectives. *Annual Review of Control, 52*, 170–196. https://doi.org/10.1016/j.arcontrol.2021.10.008

[37] Berberich, J., & Allgöwer, P. (2024). An overview of systems-theoretic guarantees in data-driven model predictive control. *Annual Review of Control, Robotics, and Autonomous Systems, 8.* https://doi.org/10.1146/annurev-control-030323-024328

[38] Mitchell, D., Blanche, J., Zaki, O., Roe, J., Kong, L., Harper, S., Robu, V., Lim, T., & Flynn, D. (2021). Symbiotic system of systems design for safe and resilient autonomous robotics in offshore wind farms. *IEEE Access, 9*, 141421–141452. https://doi.org/10.1109/ACCESS.2021.3117727

[39] Zhang, X., Li, Y., Ran, Y., & Zhang, G. (2020). Stochastic models for performance analysis of multistate flexible manufacturing cells. *Journal of Manufacturing Systems, 55*, 94–108. https://doi.org/10.1016/j.jmsy.2020.02.013

[40] Lauri, M., Hsu, D., & Pajarinen, J. (2022). Partially observable Markov decision processes in robotics: A survey. *IEEE Transactions on Robotics, 39*(1), 21–40. https://doi.org/10.1109/TRO.2022.3200138

[41] Chen, Y., Georgiou, T. T., & Pavon, M. (2021). Optimal transport in systems and control. *Annual Review of Control, Robotics, and Autonomous Systems, 4*(1), 89–113. https://doi.org/10.1146/annurev-control-070220-100858