Autonomous Navigation in Unmanned Ground Robots

Chandrajit Ganguly
Research and Development Estt (Engrs)
Defence Research and Development
Organisation
Pune – 411 015
cganguly.rde@gov.in

doi: https://doi.org/10.37285/bsp.SACAD2025.55

***Abstract*— *In robotics, autonomous navigation has become a crucial field that allows mobile robots to perceive, locate, and function autonomously in dynamic, complex, and unstructured environments. The development of autonomous navigation systems from the first rule-based machines like Shakey to the current AI-integrated, sensor-rich robotic platforms is summarized in this paper. The proposed classical process flow describes a modular pipeline that includes localization using Kalman filtering and SLAM, perception, motion control strategies (PID, Pure Pursuit), global and local path planning planners (A*, DWA, TEB) etc. To guarantee smooth system integration, the Robot Operating System (ROS) is utilized. With a special emphasis on Indian defence robotics, the paper highlights practical applications in the defence sectors. The paradigm shift towards cognitive autonomy is discussed in relation to the transformative role of artificial intelligence, specifically deep learning and reinforcement learning, even though the framework described depends on deterministic methods and optimisation-driven control. This integration promises mission-ready, scalable, and adaptable autonomous systems that can operate in unstructured and high-stakes situations.*

Keywords*— *Autonomous Navigation, Mobile Robots, SLAM, Sensor Fusion, Artificial Intelligence, Path Planning, Defence Robotics

I. Introduction

Autonomous navigation enables mobile robots to operate with minimal human intervention in complex, dynamic, and unstructured environments [1, 2]. It integrates technologies such as artificial intelligence, control systems, sensor fusion, machine learning [3, 4] for making it essential for both civilian and defence applications. Modern systems rely on rich sensor suites (e.g., 2D/3D LiDAR, GPS, cameras, IMUs) and probabilistic estimation techniques like Kalman filters and SLAM for localization and mapping [5, 6][8–10]. Navigation pipelines combine global and local planning algorithms (e.g., A*, DWA) with motion control strategies (e.g., PID, MPC) to ensure accurate motion [7, 8]. As AI advances, autonomous navigation is being deployed in vehicles, logistics robots, legged robots, industrial inspectors, and military platforms [9-13].

This paper presents a comprehensive review of autonomous navigation systems, focusing on their evolution, architecture, and real-world applications—particularly in defence and field robotics. It explores key technological advancements, highlights practical use cases across sectors like defence, industry, proposes an ideal navigation framework integrating SLAM, sensor fusion, cognition, planning, and control, and evaluates emerging challenges along with future research opportunities.

II. Review of the Development of Autonomous Navigation Systems

A. Evolution of Core Technologies

Autonomous navigation has progressed from basic Automated Guided Vehicles (AGVs) in the 1950s to AI-driven mobile platforms of today [14, 15]. Early innovations like Shakey integrated perception, planning, and actuation [3, 16][3, 6], while reactive architectures such as the subsumption model enabled real-time responsiveness in the 1980s–90s [1]. The introduction of middleware like ROS in the 2000s standardized robotic software development and fostered modular integration [17].

B. The Rise of AI-based Navigation

In the 2010s, learning-based systems began to replace rule-based models. CNNs improved perception, while reinforcement learning enabled adaptive control policies [18, 19]. Tesla's Full Self-Driving system exemplifies this shift with an end-to-end neural architecture for perception and planning [10, 20]. Legged robots like Spot and Digit also leverage AI and multi-sensor fusion for terrain adaptation [1], while platforms like Skydio drones and AUVs integrate AI for aerial and underwater navigation [7]

C. Industrial and Commercial Applications

Autonomous robots have transformed sectors such as logistics (e.g., Amazon Kiva robots) [21], agriculture (e.g., John Deere's See & Spray) [19], and healthcare (e.g., TUG and Relay service robots) [4]. These systems improve operational efficiency through fleet coordination, precision automation, and autonomous service delivery.

D. Defence Applications

In defence, autonomous systems enhance mission effectiveness in hazardous GPS-denied environments. High agile defence platforms use 2D/3D LiDAR, IMUs, stereo cameras, and stochastic filters for reconnaissance and patrol [12, 13]. Global systems like the Uran-9 tank and SMET vehicle integrate AI for tactical support and logistics under minimal supervision [24].

E. Cognitive Autonomy and Ethical Considerations

The field of autonomy is transitioning from scripted behavior to **cognitive autonomy**, with architectures like SOAR and ACT-R enabling reasoning under uncertainty [22]. Robots can now infer human intent, adapt to social settings, and operate in partially known environments [4, 8, 9]. As autonomy increases, addressing ethical and legal issues in decision-making becomes critical, especially in high-stakes domains such as defence, caregiving, and transport.

III. Idealized Steps to Develop an Autonomous Navigation Robot

In the journey toward full autonomy, a robot must evolve from a simple actuator-based device to an intelligent system capable of perceiving, interpreting, deciding, and acting—seamlessly and safely. Autonomous navigation, therefore, is not just a feature but an ecosystem of integrated modules working in harmony. Though multiple development processes exist across research and industry, yet this section outlines a holistic approach, grounded in real-world implementation and simulation-backed design. Autonomous navigation is not linear—it is a looped, multi-threaded architecture of perception, localisation, cognition, and motion control, constantly updated in real-time. Below is a step-by-step, high-fidelity framework that serves as a guide to engineer such a system.

A. Localization and Mapping: Understanding "Where Am I"?

Accurate localisation and environment mapping are fundamental to any autonomous navigation system. While path planning and control determine *how* a robot moves, localisation and mapping determine *where* it is and *what* surrounds it (Figure 1Figure 1). This section details a progression from basic dead reckoning to advanced Simultaneous Localisation and Mapping (SLAM) techniques, supported by mathematical formulations and real-world algorithmic implementations.

1) Odometry and Dead Reckoning: Robots typically begin navigation with internal motion estimates derived from wheel encoders and inertial measurement units (IMUs)[23, 24]. This process, known as dead reckoning, calculates position updates based on the previous pose and estimated motion:

$$x_t = x_{t-1} + u_t + w_t \quad (1)$$

Where, x_t is the current pose; u_t is the control input (from encoders); w_t represents system noise. While simple and fast, dead reckoning suffers from drift accumulation due to wheel slippage, surface irregularities, or encoder resolution limits. Thus, it is insufficient for long-term or accurate navigation.

2) Sensor Fusion via Kalman Filters: To enhance localisation accuracy, sensor fusion is applied using the Kalman Filter (KF) [1, 11], which statistically combines predictions from the robot's motion model with real-time sensor measurements (e.g., GPS, LiDAR, IMU) to produce an optimal pose estimate.

a) Kalman Filter Formulation (Linear Gaussian Systems):

Robots pose at time t be X_T, where $X_T=[x,y,\theta]^T$
Robots control inputs be U_T, where $U_T=\{u_0,u_1,u_2 \ldots u_T\}$
Robots Sensor measurements inputs be $Z_T=\{z_0,z_1,z_2 \ldots z_T\}$

Problem of KF based localisation is to estimate the robot pose from the control inputs U_T and observations Z_T

$$p(X_T|Z_T,U_T) \quad (2)$$

Motion Model (Prediction Step):

$$x_t = f(x_{t-1},u_t) + \epsilon, \quad (3)$$
$$\epsilon \sim \aleph(0, \sum u) \quad \text{Noise Model}$$

For differential-drive robots:

$$x_t = x_{t-1} + v \cdot \Delta t \cdot \cos(\theta_{t-1}) \quad (4)$$

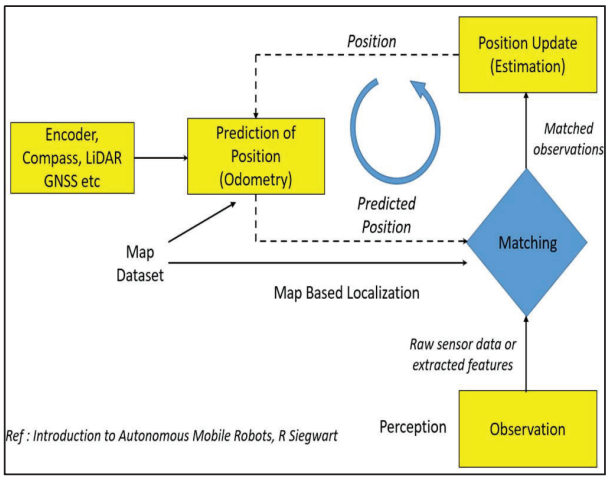

Figure 1. Concept of KF based localization

$$y_t = y_{t-1} + v \cdot \Delta t \cdot \sin(\theta_{t-1}) \quad (5)$$

$$\theta_t = \theta_{t-1} + \omega \cdot \Delta t \quad (6)$$

Defines how likely a measurement z_t is given pose:

$$p(z_t|x_t) \sim \aleph(h(x_t), \sum z) \quad (7)$$

where $h(x_t)$ is a function predicting expected measurement from pose

Kalman Filter (KF):

$$x_t = Ax_{t-1} + Bu_t + \epsilon \quad (8)$$
$$z_t = Hx_t + \delta \quad (9)$$

Where
Xt = state vector at time t
A = state transition matrix
B = Control input matrix
Ut= control vector
Zt = measurement vector at time t
$\epsilon \sim \aleph(0, Q)$ process Noise
δ = measurement noise
KF Update:

$$\text{Kalman Gain: } K_t = P_t H^T \left(HP_t H^T + R\right)^{-1} \quad (10)$$
$$x_t = x_t + K_t(z_t - Hx_t) \quad (11)$$
$$P_t = (I - K_t H)P_t \quad (12)$$

Where Kt= Kalman Gain
R = Measurement noise variance
H = observation matrix
Pt= updated error covariance

b) Extensions to Nonlinear Systems: To address the limitations of the classical Kalman Filter in nonlinear and non-Gaussian settings, advanced variants such as the Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) are employed in autonomous navigation systems [25, 26]. EKF linearises the nonlinear motion and observation models via Jacobian matrices. The **prediction step** is given by:

$$x_{k|k-1} = f(x_{k-1}, u_k), \quad P_{k|k-1} = F_k P_{k-1} F_k^T + Q_k \quad (13)$$

where $x_{k|k-1}$ is the predicted state, u_k is the control input, $F_k = \frac{\partial f}{\partial x}|_{x_{k-1}}$ is the Jacobian of the state transition function f, and Q_k is the process noise covariance matrix. The **update step** is defined as:

$$K_k = P_{k|k-1} H_k^T \left(H_k P_{k|k-1} H_k^T + R_k\right)^{-1} \quad (14)$$

$$x_{k|k}=x_{k|k-1}+K_k\left(z_k-h(x_{k|k-1})\right) \quad (15)$$
$$P_{k|k}=(I-K_kH_k)P_{k|k-1} \quad (16)$$

Here, K_k is the Kalman gain, z_k is the measurement vector, $h(x_{k|k-1})$ is the expected measurement, $H_k = \frac{\partial h}{\partial x}|_{x_{k|k-1}}$ is the Jacobian of the measurement model, and R_k is the measurement noise covariance.

In contrast, the UKF avoids linearisation altogether by using a deterministic sampling approach. Sigma points are generated as:

$$X_{k-1}^{(i)}=x_{k-1}\pm\sqrt{(n+\lambda)P_{k-1}}, \quad i=1,\ldots,2n \quad (17)$$

where x_{k-1} is the prior mean, P_{k-1} the prior covariance, n the state dimension, and λ a scaling parameter. These points are propagated through the nonlinear function:

$$x_{k|k-1}=\sum_{i=0}^{2n} W_i^{(m)} X_{I,k|k-1}^{*}, \quad (18)$$

$$P_{k|k-1}=\sum_{i=0}^{2n} W_i^{(c)} \left(X_{i,k|k-1}^{*}-x_{k|k-1}\right)\left(X_{i,k|k-1}^{*}-x_{k|k-1}\right)^T+Q_k \quad (19)$$

where $W_i^{(m)}$ and $W_i^{(c)}$ are weights for the mean and covariance. The update involves computing the predicted measurement mean $\hat{z}$, innovation covariance P_{zz}, and cross-covariance P_{xz}:

$$K_k=P_{xz}P_{zz}^{-1}, \quad x_{k|k}=x_{k|k-1}+K_k(z_k-\hat{z}),$$
$$P_{k|k}=P_{k|k-1}-K_kP_{zz}K_k^T \quad (20)$$

UKF thus offers superior estimation in strongly nonlinear systems and avoids the complexity of Jacobian calculations, making it particularly valuable in modern mobile robot localization tasks.

3) SLAM: Mapping While Moving: In unknown or GPS-denied environments, robots must localise themselves while building a map of the surroundings simultaneously. This coupled estimation of the robot's pose and the environment is known as Simultaneous Localization and Mapping (SLAM)[6, 7, 9, 27]. Unlike standard localization, SLAM does not assume a pre-existing map, making it essential for exploration missions, search and rescue, and autonomous vehicles. SLAM tries to generate the robot path and the environment map using the data gathered by the robots proprioceptive and exteroceptive sensors. It is a difficult problem as the estimated path and the extracted features are corrupted by noise.

Robots pose at time t be x_t. The robots path is :

$X_T=\{x_0,x_1,x_2\ldots x_T\}$ …x_0 is assumed to be known

Sequence of robots relative motion is:

$U_T=\{u_0,u_1,u_2\ldots u_T\}$ …u_t denotes robot's motion between time t-1 and t

Let M be the true map of the environment

$$M=\{m_0,m_1,m_2\ldots m_{n-1}\}$$

Robot measurements:

$$Z_T=\{z_0,z_1,z_2\ldots z_T\}$$

Problem of SLAM is to recover the model of the map M and the path X_T from the odometry U_T and observations Z_T:

$$p(X_T,M|Z_T,U_T) \quad (21)$$

Several practical Simultaneous Localization and Mapping (SLAM) algorithms have been developed to address different sensor configurations and computational constraints.

ORB-SLAM (Oriented FAST and Rotated BRIEF SLAM) is a feature-based SLAM system that tracks distinctive key points across image frames using ORB features [6, 27, 28]. It supports monocular, stereo, and RGB-D cameras, making it versatile for various applications. A key strength of ORB-SLAM is its robust loop closure detection, which ensures long-term map consistency by recognizing previously visited locations (Figure 2). This makes it particularly effective in structured environments—both indoor and outdoor—where distinct visual features are available [2, 25].

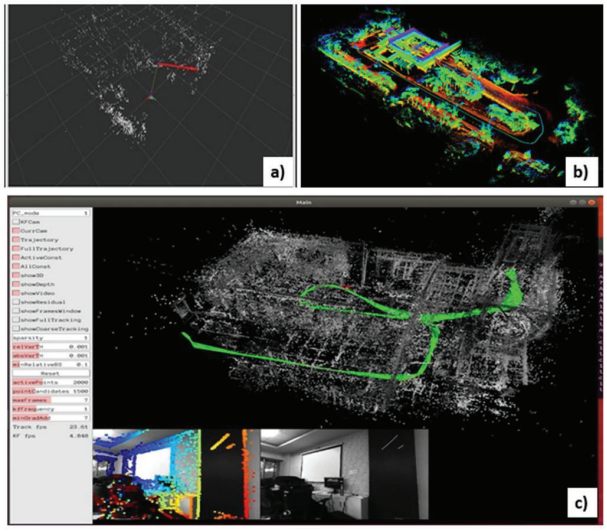

Figure 2. Integrated SLAM outputs of a). ORB SLAM; b) LeGO-LOAM and c) DSO

In contrast, DSO (Direct Sparse Odometry) bypasses traditional feature extraction by working directly on pixel intensities (Figure 2). Instead of relying on feature matching, it minimizes photometric error to estimate camera motion [4, 29]. This approach reduces computational overhead, making

DSO well-suited for low-power or embedded systems, such as drones or mobile robots, where real-time performance is critical [25].

For LiDAR-based SLAM, LeGO-LOAM (Lightweight and Ground-Optimized LiDAR Odometry and Mapping) offers an efficient solution by segmenting ground and edge features from 3D point clouds [7, 30]. This optimization improves pose estimation accuracy, particularly in outdoor environments with varying terrain, vegetation, and elevation changes (Figure 2). Its lightweight design ensures real-time performance even on resource-constrained platforms (Ge & Lewis, 2006).

Each of these algorithms—ORB-SLAM, DSO, and LeGO-LOAM—addresses different challenges in SLAM, balancing accuracy, computational efficiency, and sensor compatibility for diverse robotic applications (Gul et al., 2019; Ge & Lewis, 2006).

B. Perception Layer: Setting the World with Sensors

The first capability that separates a manual robot from an autonomous one is the ability to sense its surroundings. This is where the perception layer comes into play.

1) Sensor Selection: The choice of sensors must be aligned with the operating terrain, the accuracy required, and environmental dynamics (e.g., light variation, obstacles, and dynamic agents). The sensors broadly fall into two categories:

- Proprioceptive sensors like wheel encoders, tachometers, and IMUs provide internal motion and orientation data.
- Exteroceptive sensors such as LiDAR, cameras (mono, stereo, RGB-D), radar, ultrasonic, and GNSS perceive the external world.

To achieve reliable perception:

- Redundant sensor pairs (e.g., front and side LiDARs) provide coverage in complex terrains.
- Sensor placement—height, field of view, and vibration damping directly affect data accuracy.
- Weather-proofing and calibration are non-negotiables for deployment in the field.

2) Challenges in Sensor Selection: The performance of autonomous mobile robots depends significantly on sensor type, placement, and integration. However, sensor selection is complex due to varying environments and mission demands. Cameras work well in feature-rich areas but struggle in low light or glare, while LiDAR offers precise range data but is costly and less effective in adverse weather[31]. High-resolution sensors improve detail but increase processing load, and using multiple sensors enhances reliability but consumes more power. Practical deployments often require simpler sensor suites, compensated by intelligent fusion and filtering methods [32].

3) Multi-Sensor Data Fusion (MSDF) and its Mathematical Model: Multi-Sensor Data Fusion (MSDF) is critical for robust environmental perception in autonomous robots. This process combines inputs from various sensors to obtain more accurate and robust estimates of the environment and the robot's own state (Figure 3). By combining heterogeneous sensor inputs, MSDF improves state estimation accuracy and resilience to sensor failures. Sensor fusion helps reduce uncertainty, enhance reliability, and ensure redundancy. MSDF integrates proprioceptive sensors (e.g., wheel encoders, IMUs) with exteroceptive sensors (e.g., LiDAR, stereo cameras, RADAR, GNSS), thereby offering a consistent estimate even when some sensors fail or provide noisy data [8, 26].

To enhance reliability and robustness in the perception layer, especially within Multi-Sensor Data Fusion (MSDF), a formalized mathematical framework is essential. This fusion process integrates multiple sensor observations to reduce individual uncertainties and generate a consistent, accurate environmental model and robot state estimate.

The first component of this process is l sensor measurements. Each sensor provides a noisy observation, mathematically described as:

$$z_i = h_i(x) + v_i \quad (22)$$

Here, z_i is the measurement vector from sensor i, $h_i(x)$ is the observation function that maps the robot's actual state x to the expected sensor measurement, and v_i is the measurement noise, typically assumed to be zero-mean Gaussian with covariance R_i, i.e., $v_i \sim \mathcal{N}(0, R_i)$. This formulation captures uncertainties inherent in sensors like LiDAR (which includes range and angular noise), cameras (affected by pixel noise and lens distortion), and IMUs (subject to bias and Gaussian noise).

The second component is Kalman Filter-based sensor fusion. For nonlinear systems, the Extended Kalman Filter (EKF) is commonly applied, which linearises the nonlinear motion and measurement models using Jacobians. The prediction step is given by:

$$X_{k|k-1} = f(X_{k-1}, u_k) + w_k \quad (23)$$

In this expression, $X_{k|k-1}$ is the predicted state at time k, $f(\cdot)$ is the state transition function (e.g., IMU motion model), X_{k-1} is the prior state, u_k is the control input (e.g., IMU accelerations or angular velocities), and $w_k \sim N(0, Q_k)$ is the process noise with covariance Q_k.

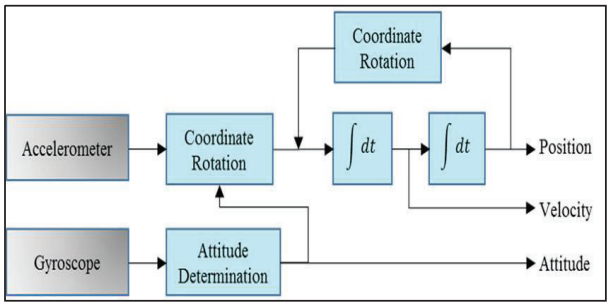

Figure 3. Generic sensor fusion model

Unlike the Extended Kalman Filter (EKF), which relies on linearizing nonlinear functions using Jacobians, the Unscented Kalman Filter (UKF) bypasses linearization by employing a set of deterministically chosen sigma points. These sigma points are propagated through the "" 'system's nonlinear dynamics to estimate the mean and covariance of the state distribution accurately. At time $k-1$, the sigma points are defined as

$$X_{k-1} = \left[x_{k-1}, \quad x_{k-1} \pm \sqrt{(n+\lambda)P_{k-1}}\right] \quad (24)$$

where $\mathbf{x}_{k-1}$ represents the prior state estimate, P_{k-1} is the associated covariance matrix, n is the dimension of the state vector, and λ is a scaling parameter that tunes the spread of the sigma points. Once the sigma points are generated, they are propagated through the system model, and the predicted mean and covariance are calculated as follows:

$$x_{k|k-1} = \sum_{i=0}^{2n} W_i^{(m)} X^*_{i,k|k-1} \quad (25)$$

$$P_{k|k-1} = \sum_{i=0}^{2n} W_i^{(c)} \left(X^*_{i,k|k-1} - x_{k|k-1}\right)\left(X^*_{i,k|k-1} - x_{k|k-1}\right)^T + Q_k \quad (26)$$

Here, $X^*_{i,k|k-1}$ denotes the propagated sigma points, $W_i^{(m)}$ and $W_i^{(c)}$ are the weights associated with the mean and covariance, respectively, and Q_k represents the process noise covariance. This method avoids the pitfalls of Jacobian-based linear approximations, making it more suitable for systems with strong nonlinearities. Although UKF is computationally more intensive than EKF, its accuracy and robustness in highly nonlinear domains make it an essential tool in autonomous navigation, especially for applications involving complex motion dynamics or unreliable sensor observations.

The third component is probabilistic fusion, grounded in Bayesian estimation. It combines the sensor likelihoods to

compute a fused posterior state estimate. This is expressed as:

$$p(X|z_1,\ldots,z_N) \propto p(X) \prod_{i=1}^{N} p(z_i|X) \quad (27)$$

where $p(X)$ is the prior probability of state X, and $p(z_i \mid X)$ is the likelihood of measurement z_i given state X. This approach allows for probabilistic handling of uncertainty from multiple, potentially conflicting, sensor inputs.

The final stage of this model is implemented effectively through Factor Graph Optimization, particularly using GTSAM (Georgia Tech Smoothing and Mapping). GTSAM solves SLAM as a maximum a posteriori (MAP) problem, optimizing a factor graph that links robot poses X^* and landmarks L^*. The optimization minimizes the sum of squared residuals:

$$X^*, L^* = \underset{X,L}{argmin} \left(\sum_{odometry} || f(X_i, u_i) - X_{i+1} ||^2_{Q_i} + \sum_{measurements} || h(X_i, l_j) - z_{ij} ||^2_{R_{ij}} \right) \quad (28)$$

This includes:

- Odometry factors: $f(X_i, u_i)$, the motion model predicting pose X_{i+1} from control input u_i with noise Q_i,
- Measurement factors: $h(X_i, l_j)$, the observation model predicting measurement z_{ij} of landmark l_j from pose X_i, with noise covariance R_{ij},
- Additional constraints: For loop closure and drift correction.

By enabling incremental updates, efficient sparse graph computation, and global consistency, GTSAM enhances real-time SLAM performance, particularly when combining visual and inertial data as seen in implementations like ORB-SLAM3 [27].

C. Cognition and Planning: Choosing "Where to Go?"

After knowing its location and surroundings, the robot must decide its path to reach a target safely and efficiently. This phase involves Global and Local Path Planning.

1) Global Path Planning: This module computes a collision-free path from the current position to a desired goal using a static/global map. These planners generate waypoints that guide the robot, but they don't account for real-time dynamics or moving obstacles.

a) Dijkstra's Algorithm: Dijkstra's algorithm computes the shortest path in a weighted graph by exploring nodes in the order of their known shortest distance from the starting point[14]. For each node v, Dijkstra's maintains:

$$dist(v) = min(dist(v), dist(u) + w(u,v)) \quad (29)$$

Here, $\text{dist}(v)$ is the minimum known distance to node v, and $w(u, v)$ is the edge weight between nodes u and v. Though it guarantees optimal results, it can be slower as it explores all directions equally. It is commonly used in network routing and robot navigation, especially when edge weights represent actual traversal cost or terrain difficulty.

b) A*: The A* algorithm finds the shortest path between two points by combining Dikjstra's cost (actual travel cost) with a heuristic (estimated cost-to-goal, e.g., Euclidean distance) [2, 14]. This makes it faster than Dijkstra's algorithm in many cases, especially when the heuristic is well-designed. A* guarantees optimal paths if the heuristic does not overestimate.

A* Algorithm prioritises nodes with the least cost:

$$f(n)=g(n)+h(n) \quad (30)$$

Where, $g(n)$: actual cost from the start node to node n, and $h(n)$: heuristic estimate of the cost from n to the goal. If the heuristic is admissible (never overestimates), A* guarantees an optimal path while being significantly faster than Dijkstra's in practice.

c) RRT (Rapidly-exploring Random Tree): RRT is designed for fast exploration of large or high-dimensional spaces, making it ideal for robotic motion planning [33]. It incrementally builds a tree by randomly sampling points and connecting them to the nearest valid point in the tree.

$$\textit{Tree growth: } q_{new} = q_{nearest} + \eta \cdot \frac{q_{random} - q_{nearest}}{\|q_{random} - q_{nearest}\|} \quad (31)$$

Where: q_{random} : randomly sampled configuration, q_{nearest} : nearest tree node, η : step size controlling the extension. Variants like RRT* improve optimality, while bi-directional versions such as RRT-Connect accelerate convergence. RRT handles complex/dynamic environments well but may planning and produce suboptimal paths. It is widely used in autonomous driving, drone navigation, and robotic arm trajectory

2) Local Path Planning: Local path planning is primarily focused on real-time obstacle avoidance while navigating toward a goal. Unlike global planners that work on a complete map, local planners operate within a '" 'robot's immediate sensing range using data from sensors like 2D/3D LiDAR, stereo cameras, ultrasonic sensors, and radar.

Several algorithms are used for local planning, including Bug algorithms, Vector Field Histogram (VFH), Curvature Velocity, Dynamic Window Approach (DWA), Timed Elastic Band (TEB), etc [34]. In this classical framework, DWA and TEB will be discussed further**.** These planners are essential for dynamic and partially known environments, enabling safe and efficient robot navigation through continuous sensor feedback and motion adjustment.

a) DWA (Dynamic Window Approach): DWA is a real-time local path planning method (Figure 4) used for collision-free navigation of mobile robots [34]. It evaluates a set of admissible velocity commands (v,ω) constrained within a feasible dynamic window defined by the '" 'robot's acceleration limits and current velocity. The admissible velocities lie within specified bounds:

$$v \in [v_{min} v_{max}] \quad and \quad \omega \in [\omega_{min} \omega_{max}] \quad (32)$$

Additionally, to ensure safe stopping, velocities are limited by braking distance:

$$v \leq \sqrt{2.dist(v,\omega).a_{max}} \text{ and } \omega \leq \sqrt{2.dist(v,\omega).\alpha_{max}} \quad (33)$$

where $dist(v,\omega)$: Distance to nearest obstacle; a_{max}, α_{max}: Max linear/angular accelerations Among all feasible velocities, DWA selects the optimal pair by smaximising an objective function that balances goal heading, obstacle clearance, and forward velocity:

$$G(v,\omega) = \alpha \cdot heading(v,\omega) + \beta \cdot clearance(v,\omega) + \gamma \cdot velocity(v,\omega) \quad (34)$$

where, heading: alignment to goal, clearance: distance to obstacles, velocity: forward speed, α, β, γ: weights. This scoring mechanism enables the robot to make smooth, responsive decisions at each control cycle, maintaining both safety and efficiency during motion planning [35, 36]

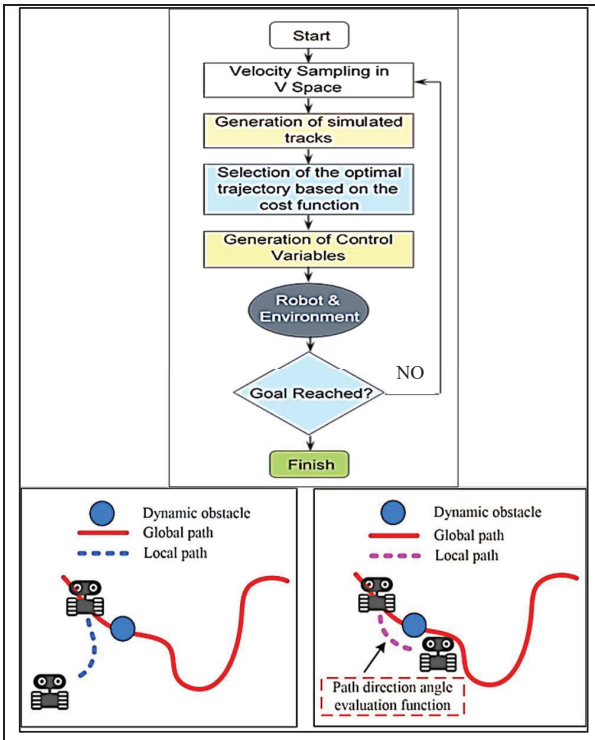

Figure 4 Dynamic Window

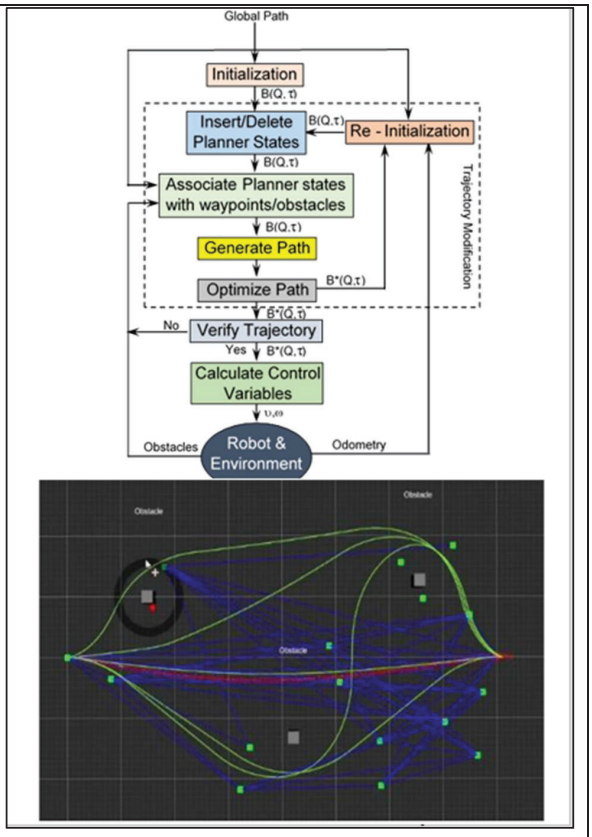

Figure 5 Timed Elastic Band

b) Timed Elastic Band (TEB): TEB algorithm (Figure 5) is a real-time local path planner that formulates trajectory generation as an optimisation problem in both spatial and temporal domains [36]. A trajectory in TEB is represented as a sequence of robot poses and time intervals:

$$B = \{x_i, \Delta T_i\}_{i=0}^{N}, \text{where } x_i = (x_i, y_i, \theta_i) \quad (35)$$

The algorithm minimizes a multi-objective cost function that balances path length, time efficiency, and obstacle avoidance:

$$J(B) = \sum_k \lambda_k \left(\|x_{i+1} - x_i\|^2 + \Delta T_i^2 + exp\left(-\frac{d(x_i,O)}{\sigma}\right)\right) \quad (36)$$

Here, $d(x_i, 0)$ is the distance from pose x_i to the nearest obstacle, and σ is a scaling parameter. To ensure feasibility,

TEB enforces smoothness by penalizing abrupt curvature changes using the constraint:

$$\|x_{i+1} - 2x_i + x_{i-1}\| \leq \epsilon \quad (37)$$

Additionally, kinematic feasibility is maintained through:

$$v_i \leq v_{max}, \omega_i \leq \omega_{max} \quad (38)$$

By optimizing this time-parameterized trajectory while considering obstacle proximity and motion constraints, TEB produces smooth and safe motion plans suitable for dynamic and cluttered environments.

B. *Motion Control: Acting with Precision*

The motion control module ensures that the robot accurately follows the trajectory planned by the global and local planners.

It serves as the final execution layer, converting high-level navigation commands into low-level actuation signals that drive the robot's motors [1, 5]. This process requires precise modelling of the robot's motion capabilities and real-time adjustments based on environmental feedback. To achieve this, motion control involves the a kinematic model, a tracking controller that guides the robot along the planned trajectory, and a real-time execution strategy that adapts to terrain and

actuator-level dynamics [2]. The following section deals with these components.

1) Kinematic Model for Differential Drive: Differential drive configurations are standard in mobile ground robots due to their simplicity and manoeuvrability [11, 37]. This setup includes two independently powered wheels on a shared axis, and their velocities determine the robot's trajectory. The linear velocity $\boldsymbol{v}$ m/s and angular velocity $\boldsymbol{\omega}$ rad/s of the robot are derived from the velocities of the left (v_l) and right (v_r) wheels as:

$$v = \frac{v_r+v_l}{2}, \quad \omega = \frac{v_r-v_l}{L} \tag{39}$$

where L is the track width—the distance between the wheels. The continuous-time pose dynamics of the robot, characterised by its position (x, y) and orientation θ, are defined as:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega \tag{40}$$

This system of differential equations can also be represented in matrix form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{41}$$

For practical implementation, especially in simulations or real-time odometry updates, the discrete-time version is computed using Euler integration over a small time step Δt:

$$\begin{aligned} x_{t+1} &= x_t + v\cos\theta_t \cdot \Delta t, \\ y_{t+1} &= y_t + v\sin\theta_t \cdot \Delta t, \\ \theta_{t+1} &= \theta_t + \omega \cdot \Delta t \end{aligned} \tag{42}$$

2) Tracking using Pure Pursuit: The **Pure Pursuit controller** is a geometric path-tracking algorithm used in mobile robots to follow a planned trajectory. It calculates a **look-ahead point** on the path and determines the required curvature for the robot to reach that point smoothly [1, 11]. By continuously updating this target as the robot moves, Pure Pursuit enables responsive and stable path tracking, making it well-suited for real-time applications in differential-drive and car-like robots. It is a popular geometric path-tracking algorithm that guides the robot toward a dynamically selected look-ahead point on the desired path. This target is updated in real time, enabling responsive, stable path following suitable for both differential-drive and car-like robots.

To initiate the process, the controller identifies a look-ahead point (x_L, y_L) that is at a predefined distance L from the robot's current position (x_r, y_r). This is mathematically expressed as:

$$\sqrt{(x_L - x_r)^2 + (y_L - y_r)^2} = L \tag{43}$$

The curvature γ of the arc connecting the robot's current position to the look-ahead point is then computed as:

$$\gamma = \frac{2\cdot\sin(\alpha)}{L} \tag{44}$$

where α is the angle between the '" 'robot's heading and the line joining its position to the look-ahead point:

$$\alpha = \text{atan2}(y_L - y_r, x_L - x_r) - \theta_r \tag{45}$$

This curvature is used to compute the differential drive wheel velocities based on a desired linear velocity v_{cmd}:

$$v_l = v_{\text{cmd}}\left(1 - \frac{\gamma\cdot L_w}{2}\right), \quad v_r = v_{\text{cmd}}\left(1 + \frac{\gamma\cdot L_w}{2}\right) \tag{46}$$

where L_w is the wheelbase (distance between the left and right wheels).

As the robot moves, its updated velocities are:

$$\dot{x}_r = v_{\text{cmd}}\cos\theta_r, \quad \dot{y}_r = v_{\text{cmd}}\sin\theta_r, \quad \dot{\theta}_r = \gamma v_{\text{cmd}} \tag{47}$$

This formulation allows real-time path adjustment while preserving smoothness and stability in the robot's trajectory.

3) Real-Time Execution: Real-time execution in motion control ensures that the robot follows the planned path accurately while adapting to dynamic conditions. PID controllers are commonly used to adjust torque and velocity, correcting deviations between desired and actual motion. Effective performance requires gain tuning that is specific to terrain type—for example, lower gains on slippery surfaces[5]. Additionally, the control loop must account for practical factors such as tire slip, actuator lag, and terrain resistance to maintain stability and responsiveness during operation.

C. System Integration: Connecting the Dots in ROS

In a fully autonomous robot, the seamless interaction of perception, localization, planning, and control modules is essential. This integration is achieved using the Robot Operating System (ROS), a modular middleware framework that enables communication between various software nodes via publish–subscribe mechanisms and service calls.

1). Core ROS Nodes and Topics:

- /odom: Publishes odometry data based on wheel encoders or sensor fusion, representing the '" 'robot's estimated position and velocity.
- /tf: Maintains a dynamic transformation tree between coordinate frames (e.g., base_link, map, odom), allowing spatial alignment of sensor data.
- /scan: Streams LiDAR or laser scan data, critical for obstacle detection and local planning.
- /cmd_vel: Accepts velocity commands from local planners (e.g., DWA, TEB), translating them into actuator-level signals.

2). Configuration and Parameter Management: Planner behaviour and environmental representation are defined through configuration files such as costmap_params.yaml, which specify parameters like:

- Inflation radius: The buffer distance around obstacles,
- Obstacle persistence: The duration for which obstacles remain in the costmap,
- Robot footprint, resolution, and sensor update rates.
- These files ensure consistent behaviour across simulations and deployments and allow fine-tuning for environment-specific conditions.

3). Visualisation and Dynamic Tuning: ROS provides powerful visualisation and debugging tools for real-time system analysis:

- rqt_graph: Displays active nodes and their communication links, helping verify message flow and dependencies.
- rqt_reconfigure: Enables live adjustment of parameters such as DWA velocity limits, look-ahead distances, or TEB time horizons without restarting nodes.

Such tools are critical for testing, validating, and refining performance in both simulated and real-world scenarios.

D. Semantic Perception: Understanding, Not Just Seeing

Beyond basic obstacle detection, modern autonomous robots must semantically understand their environment to make context-aware decisions. This involves not just identifying the presence of an object, but classifying what it is—for example:

- Is the object a traffic cone, a human, or a wall?
- Is the path ahead a road lane or a pedestrian sidewalk?

Such understanding is achieved through semantic segmentation, a computer vision technique in which each pixel of an image is assigned a semantic class label. Popular deep learning models used for this task include U-Net, FastFCN, DeepLabV3+, and Mask R-CNN. These models enable fine-grained scene interpretation, which, after post-processing, translates into actionable map regions. This facilitates enhanced behavior planning, safer navigation, and improved social compliance in human-centric environments.

E. Simulation and Testing

Before real-world deployment, autonomous navigation systems undergo extensive simulation and testing to ensure reliability and performance. Tools like Gazebo are used for physics-based simulation, while Rviz aids in visualizing sensor data and robot behavior. For detailed modelling of kinematics and terrain interactions, platforms such as MATLAB, Recurdyn, and ANSYS are employed. Field trials further validate system robustness by evaluating parameters such as speed limits under autonomous and manual modes, real-time obstacle avoidance, dynamic re-routing capabilities, SLAM loop closure accuracy, and wheel slip corrections under varying terrain conditions. Following figures show some of the field trials results of an Unmanned Ground Vehicle developed on the above discussed theory. Fig 6 and Fig 7 shows the autonomous navigation of the UGV when the goal was right at the front of the UGV and some static obstacles were placed on the path. It can be seen that the UGV generated the Global straight Line path as shown in Green arrows, the local planner as red lines. While it encountered the obstacles, the local planner (TEB) generated optimal obstacle avoidance path to avoid the obstacles and after successful avoidance the UGV came back to the original global path and moved towards goal. Figure 8 shows a difficult path planned as figure of 8 where the UGV needs to change its translational and rotational velocity to track the path keeping the UGV on the commanded path with least error.

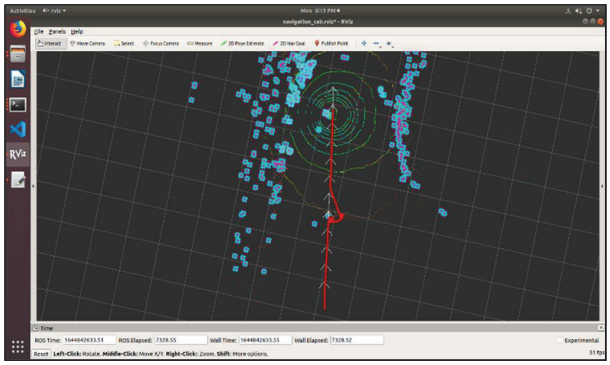

Figure 6 Autonomous UGV avoiding obstacles

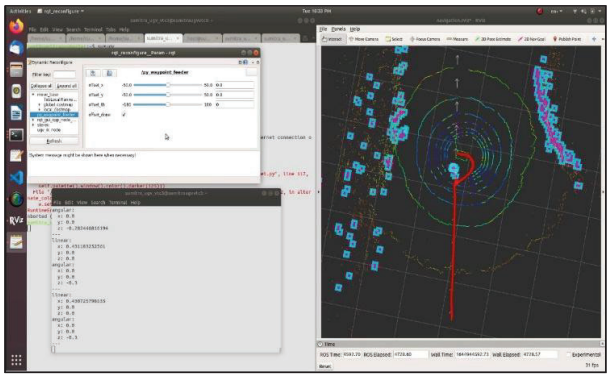

Figure 7 Autonomous UGV avoiding obstacles on the move

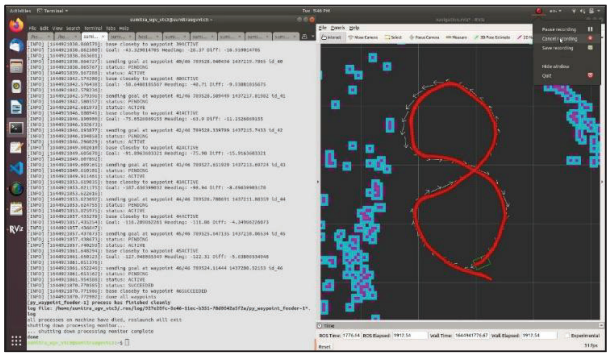

Figure 8 Autonomous UGV performing Figure of 8

F. Dynamic Tuning for Terrain Adaptability

A ground robot navigating on an asphalt surface behaves differently from one on loose sand, even though everything remains same. Hence, to cater for these types of unmodelled dynamics:

- Control gains must be dynamically tunable.
- IMUs should be hard iron, soft iron and temperature-calibrated.
- GNSS-INS units should ideally use dual antennas with tight MEMS coupling.
- Encoders must be high-resolution for better odometry.
- Wheel RPM offsets must be corrected dynamically in software.

The process of building an autonomous navigation robot is as much about integration as it is about innovation. From

accurate localization and adaptive planning to real-time control and semantic understanding, each module must feed the next with reliability and minimal delay. When designed properly, these robots can operate across varying terrains and contexts with a high degree of autonomy, safety, and intelligence.

II. CONCLUSION

Focusing primarily on ground-based mobile robots, this paper offers a thorough synthesis of the technological development, present capabilities, and architectural composition of autonomous navigation systems. The paper conceptualises how deterministic control systems have paved the way for more autonomous agents by tracing the trajectory from early Automated Guided Vehicles (AGVs) and the seminal Shakey project to the emergence of modular, sensor-integrated platforms empowered by SLAM, Kalman filtering, and layered path planning architectures. Grounded in traditional robotics ideas, a classical process flow was created and shown comprising important subsystems including perception, localization, planning, and motion control as well as implementation techniques using the Robot Operating System (ROS).

The review of real-world applications—spanning logistics, healthcare, agriculture, and defence—highlights the maturity of conventional approaches and their scalability across several domains. The study, meanwhile, also questions their shortcomings in very dynamic or uncertain settings. The conversation emphasizes that although deterministic models provide stability and clarity, their reliance on pre-defined rules, physical approximations, and handcrafted characteristics often limits them. Now pushing a paradigm change towards cognitive autonomy is artificial intelligence (AI), especially deep learning and reinforcement learning. These methods let robotic systems learn straight from data, adjust to new situations, and carry out more complicated real-time decision-making tasks. The combination of artificial intelligence promises significant gains in contextual awareness, predictive planning, and fault tolerance—qualities absolutely vital for mission-critical applications in sectors including national defence and autonomous logistics.

The implementation of AI-augmented autonomous navigation systems presents transforming possibilities for nations like India, where strategic, industrial, and humanitarian needs intersect with difficult terrain and constraints such as limited funding, infrastructure gaps, bandwidth and connectivity challenges, and shortages of skilled technical manpower.. Their inclusion into real-time, embedded robotic platforms will signal a clear move towards fully autonomous, resilient, and intelligent systems as AI models grow more efficient and generalizable.

Future research should emphasize creating hybrid architectures that mix the adaptability of learning-based systems with the strength of classical models. Operating across many different and demanding situations, next-generation robotic platforms will be more autonomous, safe, and efficient because of this convergence.

ACKNOWLEDGMENT

The author would like to thank Director RDE and Dr. S.E Talole, Sc “H” for their valuable input and support during the course of this work. The author also thanks Research And Development Establishment (Engrs), Pune and Centre for Artificial Intelligence and Robotics, Bengaluru for providing the necessary infrastructure and resources.

REFERENCES

[1] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Bradford Company, 2004.

[2] S. S. Ge and F. L. Lewis, *Autonomous mobile robots :sensing, control, decision-making, and applications* (Control engineering ; 22). Boca Raton, FL: CRC/Taylor & Francis, 2006.

[3] L. Yang *et al.*,“Path Planning Technique for Mobile Robots: A Review,” vol. 11, no. 10, p. 980, 2023.

[4] A. Elfes,“Using occupancy grids for mobile robot perception and navigation,” *Computer,* vol. 22, no. 6, pp. 46-57, 1989.

[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[6] R. Mur-Artal and J. D. Tardós,“ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras,” *IEEE Transactions on Robotics,* vol. 33, no. 5, pp. 1255-1262, 2017.

[7] T. Shan and B. Englot, *LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain*. 2018, pp. 4758-4765.

[8] P. Biber and T. Duckett, *Dynamic Maps for Long-Term Operation of Mobile Service Robots*. 2005, pp. 17-24.

[9] J. Zhang and S. Singh,“LOAM : Lidar Odometry and Mapping in real-time,” *Robotics: Science and Systems Conference (RSS),* pp. 109-111, 01/01 2014.

[10] Tesla Inc.,“Full Self-Driving (FSD),” in “Tesla AI Software Documentation,” 2022.

[11] T.-S. Su, S.-S. Lee, W.-H. Hsu, and S.-H. Fu,“A fuzzy-based approach to improve the human pick-to-light efficiency incorporated with robots behavior in an intelligent distribution center,” *Procedia Manufacturing,* vol. 38, pp. 776-783, 2019/01/01/ 2019.

[12] A. B. Rashid, A. K. Kausik, A. Al Hassan Sunny, and M. H. Bappy,“Artificial Intelligence in the Military: An Overview of the Capabilities, Applications, and Challenges,” vol. 2023, no. 1, p. 8676366, 2023.

[13] U. Matthew, J. Kazaure, A. Onyebuchi, O. Okey, I. Muhammed, and N. Okafor, *Artificial Intelligence Autonomous Unmanned Aerial Vehicle (UAV) System for Remote Sensing in Security Surveillance*. 2021, pp. 1-10.

[14] Z. Jiang, W. Wang, W. Sun, and L. Da, “Path Planning Method for Mobile Robot Based on a Hybrid Algorithm,” *Journal of Intelligent & Robotic Systems,* vol. 109, no. 3, p. 47, 2023/10/26 2023.

[15] A. Gutierrez and R. Barber, *Mobile robots history*. 2005.

[16] R. Brooks,“A robust layered control system for a mobile robot,” *IEEE Journal on Robotics and Automation,* vol. 2, no. 1, pp. 14-23, 1986.

[17] M. Quigley *et al.*, *ROS: an open-source Robot Operating System*. 2009.

[18] C. Zhang, P. Cheng, B. Du, B. Dong, and W. Zhang,"AUV path tracking with real-time obstacle avoidance via reinforcement learning under adaptive constraints," *Ocean Engineering,* vol. 256, p. 111453, 2022/07/15/ 2022.

[19] J. Yuan, H. Wang, H. Zhang, C. Lin, D. Yu, and C. Li,"AUV Obstacle Avoidance Planning Based on Deep Reinforcement Learning," vol. 9, no. 11, p. 1166, 2021.

[20] P. Mirowski *et al.*,"Learning to Navigate in Complex Environments," 11/11 2016.

[21] C. Yan, X. Xiaojia, and C. Wang,"Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments," *Journal of Intelligent & Robotic Systems,* vol. 98, 05/01 2020.

[22] E. C. Tolman,"Cognitive maps in rats and men," *Psychological Review,* vol. 55, no. 4, pp. 189-208, 1948.

[23] J. Engel, V. Koltun, and D. Cremers,"Direct Sparse Odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 40, no. 3, pp. 611-625, 2018.

[24] H. Yang, Y. Ge, Y. Shi, and H. Fang, *RA-LIO: A Robust Adaptive Tightly-Coupled Lidar-Inertial Odometry*. 2024, pp. 3863-3869.

[25] F. Gul, R. Wan, and S. S. and Nazli Alhady,"A comprehensive study for robot navigation techniques," *Cogent Engineering,* vol. 6, no. 1, p. 1632046, 2019/01/01 2019.

[26] J. Cheng, Y. Lu, E. Thomas, and J. Farrell,"Data Fusion via Kalman Filter: GPS and INS"""""," 2018, pp. 99-148.

[27] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós,"ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM," *IEEE Transactions on Robotics,* vol. 37, no. 6, pp. 1874-1890, 2021.

[28] S. Shao,"A Monocular SLAM System Based on the ORB Features," in *2023 IEEE 3rd International Conference on Power, Electronics and Computer Applications (ICPECA)*, 2023, pp. 1221-1231.

[29] I. Hussain, X. Han, and J.-W. Ha,"Stereo Visual Odometry and Real-Time Appearance-Based SLAM for Mapping and Localisation in Indoor and Outdoor Orchard Environments," vol. 15, no. 8, p. 872, 2025.

[30] G. Xue, J. Wei, R. Li, and J. Cheng,"LeGO-LOAM-SC: An Improved Simultaneous sLocalisation and Mapping Method Fusing LeGO-LOAM and Scan Context for Underground Coalmine," *Sensors,* vol. 22, p. 520, 01/11 2022.

[31] B. Elly and A. Ahsun,"Autonomous Navigation in Dynamic Environments: A Comparative Study of AI-Driven Robotics Approaches," 11/28 2024.

[32] J. T. Licardo, M. Domjan, and T. Orehovački,"Intelligent Robotics—A Systematic Review of Emerging Technologies and Trends," vol. 13, no. 3, p. 542, 2024.

[33] S. Karaman and E. Frazzoli,"Sampling-based algorithms for optimal motion planning," vol. 30, no. 7, pp. 846-894, 2011.

[34] K. Gong, Z. Xu, and X. Zhang,"Bounded-DWA: An Efficient Local Planner for Ackermann-driven Vehicles on Sandy Terrain," in *2023 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2023, pp. 632-637.

[35] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots*. 1999, pp. 343-349.

[36] M. M. S. S, N. A. Wahab, M. S. A. Mahmud, H. Alqaraghuli, E. Samsuria, and M. Z. Romdlony,"Efficient Autonomous Navigation in Dynamic Environments: Algorithm Evaluation and Multi-Robot Coordination," in *2024 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, 2024, pp. 433-438.

[37] Y. Ou, Y. Cai, Y. Sun, and T. Qin,"Autonomous Navigation by Mobile Robot with Sensor Fusion Based on Deep Reinforcement Learning," vol. 24, no. 12, p. 3895, 2024.