Dynamic Window Approach for Path Planning in Differential Drive Robots

1. Introduction

Autonomous mobile robots need to navigate environments safely while avoiding obstacles and reaching their goals. This process requires **path planning algorithms** that balance efficiency, safety, and real-time responsiveness. The **Dynamic Window Approach (DWA)** is a popular local path planning method widely used in differential drive robots, particularly those with LiDAR sensors.

This report presents a comprehensive understanding of the DWA algorithm, from motion modeling to trajectory evaluation, including kinematic derivations and simulation principles using ROS and Gazebo.

2. What is Path Planning?

Path planning involves generating a collision-free and feasible path for a robot from its **start position** to a **goal position**, while navigating through an environment filled with obstacles.

2.1 Global vs. Local Path Planning

- **Global Path Planning**: involves computing the entire path from the start position to the goal position using a known, complete map of the environment before the robot starts moving.

  Features:

  - Uses a full map: Assumes that the layout of the environment (walls, rooms, obstacles) is already known.
  - Static planning: The plan is made once and doesn't adapt to changes unless explicitly updated.
  - Ignores dynamics: The robot is treated as a point - it assumes the robot can move in any direction instantly, which is not realistic for actual robots.

  Examples of Algorithms:

  - A* (A-Star): Finds the shortest path on a grid.
  - Dijkstra's Algorithm: Finds the lowest-cost path in a weighted graph.
  - RRT (Rapidly-Exploring Random Tree): Good for complex, high-dimensional spaces (like robotic arms).

  Example Scenario: Imagine a warehouse robot with access to a detailed map of all shelves and aisles. It uses A* to calculate the shortest route from its dock to a specific rack. This is a global path that assumes all shelves are stationary and the map won't change.

- **Local Path Planning**: Local path planning works in real-time, using data from the robot's onboard sensors (like LIDAR or cameras) to avoid unexpected obstacles and make short-term decisions.

  Features:

  - Reactive and dynamic: Continuously updates based on new sensor data.
  - Accounts for robot kinematics: Considers speed, turning radius, acceleration limits.
  - Collision avoidance: Handles dynamic environments with moving people or obstacles.
  - Short horizon planning: Plans for a few seconds or meters ahead, not the entire path.

  Example Algorithm:

  - DWA (Dynamic Window Approach): Predicts multiple short-term trajectories and picks the safest and most goal-oriented one.

  Example Scenario: The same warehouse robot is on its way to the rack, but suddenly a human walks into its path. The robot's LIDAR detects the person, and DWA recalculates a new short-term trajectory to safely go around them without stopping the entire operation.

3. Robot Model and Kinematics

3.1 Differential Drive Robot

The robot under consideration is a differential drive mobile robot, which is one of the most widely used configurations in mobile robotics. This type of robot has the following characteristics:

- Two independently driven wheels: Each wheel is equipped with its own motor, allowing independent control of its speed and direction (forward or backward).
- No lateral movement: The robot cannot move sideways; all motion is the result of a combination of forward/backward movement and rotation.

The robot's motion can be controlled using two parameters:

- Linear velocity v (in meters per second): This controls the speed at which the robot moves forward or backward.
- Angular velocity ω (in radians per second): This controls how fast the robot rotates (turns) about its center.

Because of its motion constraints (it moves forward and rotates, but cannot strafe sideways), the robot's behaviour can be accurately modelled using a simplified model known as the unicycle model.

3.2 Unicycle Model Explanation

The unicycle model is a kinematic model commonly used for differential drive robots. In this model, the robot is treated as a point with a heading direction, similar to how a unicycle or a bicycle might move.

Key Assumptions of the Unicycle Model:

- The robot moves only in the direction it is currently facing.
- The robot cannot move sideways (no lateral or strafing movement).
- The control inputs to the robot are:
    - Linear velocity v: controls forward/backward motion.
    - Angular velocity ω: controls rotation or change in heading.

Position and Orientation Variables:

Let the robot's pose (position and orientation) be defined as:

- x, y: Cartesian coordinates of the robot's position in the 2D plane.
- θ: Heading angle of the robot with respect to a fixed world frame (orientation in radians).

3.3 Derivation of Kinematic Equations

To determine how the robot moves over time, we derive the discrete-time kinematic equations based on geometry and motion constraints.

Consider a small-time step Δt during which the robot moves forward with linear velocity v, and rotates with angular velocity ω.

Displacements:

- Change in x:

$$\Delta x = v \cdot \cos(\theta) \cdot \Delta t$$

- Change in y:

$$\Delta y = v \cdot \sin(\theta) \cdot \Delta t$$

- Change in heading θ:

$$\Delta \theta = \omega \cdot \Delta t$$

Using these, we update the robot's pose for the next time step $t + \Delta t$ as:

$$x_{t+\Delta t} = x_t + v \,.\cos(\theta_t)\,.\,\Delta t$$

$$y_{t+\Delta t} = y_t + v \,.\sin(\theta_t)\,.\Delta t$$

$$\theta_{t+\Delta t} = \theta_t + \omega \,.\Delta t$$

These equations form the discrete-time unicycle motion model. They allow us to predict the future position and orientation of the robot given its current state and control inputs.

3.4 Importance in Path Planning

These kinematic equations are crucial for local path planning algorithms such as the Dynamic Window Approach (DWA). In DWA, the robot simulates multiple future trajectories by applying various combinations of v and ω over a short time horizon. Each of these simulated trajectories is computed using the above motion model.

Because the unicycle model reflects the actual movement constraints of the robot (as opposed to point-based ideal models), it ensures that the planned paths are physically feasible and executable on the robot platform.

4. Dynamic Window Approach (DWA)

4.1 Overview

The Dynamic Window Approach (DWA) is a real-time local path planning algorithm used in mobile robotics to navigate dynamic and partially known environments. Unlike global planners, which rely on a complete map, DWA focuses on generating safe, executable, and goal-directed trajectories using only the robot's current state and sensor data.

Key Concepts of DWA:

- The robot continuously evaluates a set of possible velocity commands.
- For each candidate velocity pair (v, ω), it simulates the corresponding forward trajectory over a short time horizon (typically 1–3 seconds).
- Each trajectory is then scored based on a cost function that considers:
    - Proximity to obstacles (safety)
    - Distance to the goal (goal-directedness)
    - Speed (efficiency)
- The best-scoring trajectory is chosen and executed for the current time step.
- The process repeats at every control cycle, enabling reactive, real-time navigation.

4.2 The Dynamic Window

Robots cannot change their velocities instantaneously due to physical constraints on acceleration. The dynamic window is a subset of the full velocity space that represents the set of admissible velocities that the robot can reach within a short time interval T, given its current velocity and acceleration limits.

Parameters:

- v_0, ω_0: Current linear and angular velocities of the robot
- a_{max}: Maximum linear acceleration (m/s²)
- α_{max}: Maximum angular acceleration (rad/s²)
- T: Time duration for prediction (s)

Computed Velocity Ranges:

$$v \in [\, v_0 - a_{max} \,.\, T, v_0 + a_{max} \,.\, T \,]$$

$$\omega \in [\, \omega_0 - \alpha_{max} \,.\, T, \omega_0 + \alpha_{max} \,.\, T \,]$$

This defines a velocity window in the (v, ω) space, which bounds the possible motion commands for the next planning step.

Note: This window ensures that all chosen velocities are kinematically feasible and safe to execute on the actual robot.

4.3 Trajectory Simulation

For each pair of (v, ω) within the dynamic window, DWA simulates the robot's motion using the unicycle kinematic model discussed earlier. The simulation proceeds in discrete time steps Δt over a short time horizon T, producing a trajectory composed of predicted positions and orientations.

Kinematic Equations for Forward Simulation:

$$x_{i+1} = x_i + v \,.\, \cos(\theta_i) \,.\, \Delta t$$

$$y_{i+1} = y_i + v \,.\, \sin(\theta_i) \,.\, \Delta t$$

$$\theta_{i+1} = \theta_i + \omega \,.\, \Delta t$$

These equations are iteratively applied starting from the current robot pose, generating a predicted path for the robot.

5. Trajectory Evaluation (Cost Functions)

Once all feasible trajectories are simulated using the robot's motion model, the **Dynamic Window Approach (DWA)** evaluates each one to determine the most suitable path. This evaluation uses a **cost function** that balances the goals of reaching the destination, maintaining safety, and optimizing motion efficiency.

5.1 Multi-Objective Cost Function

Each trajectory is scored using a weighted sum of three criteria:

$$G(v, \omega) = \alpha \,.\, G_{heading} + \beta \,.\, G_{clearance} + \gamma \,.\, G_{velocity}$$

Where:

- $G_{heading}$: Measures how well the trajectory points toward the goal.
- $G_{clearance}$: Assesses how far the path stays from obstacles (higher = safer).
- $G_{velocity}$: Rewards higher forward speed.

Each term is scaled by a weight (α, β, γ) that must be **tuned based on the robot's operating environment** and mission priorities.

5.2 Explanation of Components

1. Heading Cost (Goal Alignment)

This component penalizes trajectories that veer away from the goal:

$$G_{heading} = -\sqrt{(x - x_g)^2 + (y - y_g)^2}$$

Where:

- (x, y): End position of the simulated trajectory
- (x_g, y_g) : Target (goal) position

The smaller the distance to the goal, the less negative this value is—hence more desirable.

2. Clearance Cost (Obstacle Avoidance)

This evaluates how close the trajectory comes to any obstacle. Trajectories that stay farther from obstacles receive higher scores:

$$G_{clearance} = min(distance\ to\ nearest\ obstacle\ along\ the\ path)$$

This ensures safety and reduces collision risk.

3. Velocity Cost (Speed Preference)

This term rewards faster motions when safe:

$$G_{velocity} = v$$

Where vvv is the linear velocity. Encouraging higher speeds helps improve motion efficiency, particularly in open environments.

5.3 Braking Distance Check (Safety Constraint)

Before considering a velocity safe, DWA includes a **braking distance** check to ensure that the robot can come to a stop without hitting any obstacle:

$$d_{brake} = \frac{v^2}{2 \, . \, a_{max}}$$

Where:

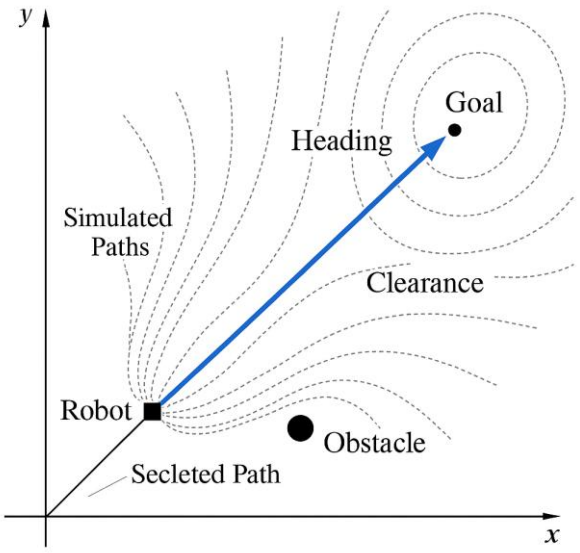
- v: Linear velocity
- a_{max}: Maximum linear deceleration

If any obstacle is closer than d_{brake}, the trajectory is discarded as unsafe.

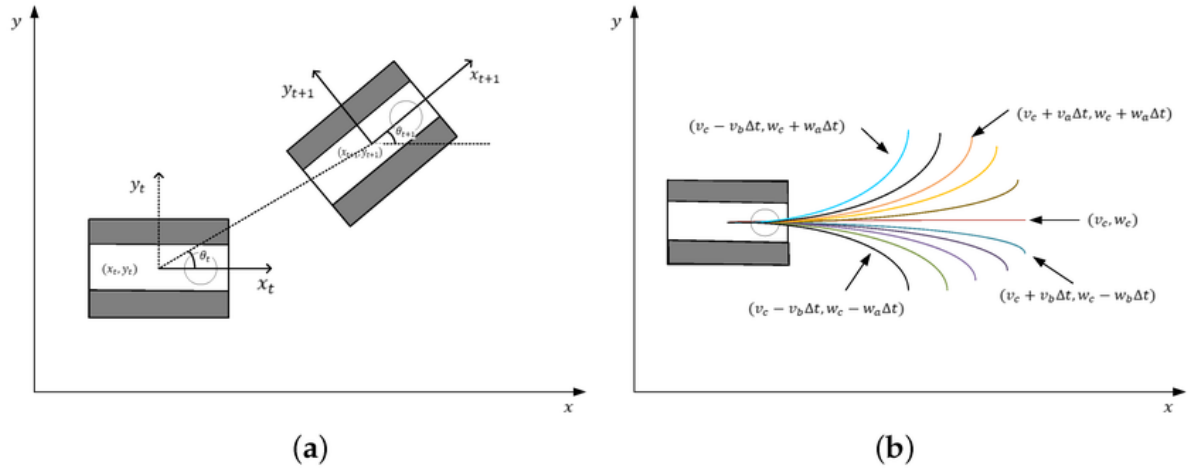
5.4 Example

Below is a diagram illustrating how cost functions influence trajectory selection:

- Trajectories closer to the goal receive better heading scores.
- Those keeping a safe distance from nearby obstacles are scored higher in clearance.
- Faster trajectories are prioritized only when safety is ensured.



Trajectory Evaluation



(a) Motion trajectory analysis diagram of electric logistics vehicle; (b) Speed sampling space.

Here is a detailed step-by-step explanation of the **Dynamic Window Approach (DWA) Algorithm** suitable for a technical report, along with a visual representation:

6. DWA Algorithm: Step-by-Step

The Dynamic Window Approach (DWA) is a real-time local path planning algorithm that selects the most suitable velocity commands for a mobile robot based on its current state, dynamics, and surrounding obstacles. Below are the core steps of the algorithm:

Step 1: Get the Current State

Capture the current state of the robot:

- Position: (x, y)
- Orientation: θ
- Linear velocity: v
- Angular velocity: ω

These state parameters are essential for simulating possible trajectories and evaluating their feasibility in real time. The required information is typically derived from the following onboard systems:

1. Odometry

Odometry refers to the estimation of the robot's position and orientation based on wheel encoder data. It calculates the displacement of the robot by measuring the rotation of its wheels over time. This method is widely used due to its simplicity and low computational cost. However, odometry is susceptible to cumulative errors caused by wheel slippage, uneven terrain, or mechanical imperfections.

2. Inertial Measurement Unit (IMU)

An IMU is a sensor device that measures the robot's angular velocity and linear acceleration. It typically contains a combination of accelerometers and gyroscopes. The angular velocity data is especially useful for determining changes in orientation, while acceleration can be integrated over time to estimate changes in velocity. Although IMUs provide high-frequency motion updates, they tend to drift over time and therefore require fusion with other data sources to maintain accuracy.

3. Localization Systems

Localization systems combine data from various onboard sensors to estimate the robot's global pose. These systems may use sensor fusion algorithms such as the Extended Kalman Filter (EKF) or Monte Carlo Localization (MCL). Advanced localization techniques may also employ Simultaneous Localization and Mapping (SLAM), which enables the robot to map its environment while simultaneously estimating its position within that map. These systems offer significantly improved accuracy over raw odometry or IMU data alone.

Step 2: Compute the Dynamic Window

Using the robot's current velocity and its acceleration limits, calculate the **dynamic window**, which represents the set of possible velocity pairs (v, ω) the robot can achieve in the next time interval Δt.

$$v \in [\, v_0 - a_{max} \,.\, T, v_0 + a_{max} \,.\, T \,]$$

$$\omega \in [\, \omega_0 - \alpha_{max} \,.\, T, \omega_0 + \alpha_{max} \,.\, T \,]$$

This step ensures that the robot only considers motions that are physically feasible.

Step 3: Simulate Trajectories

For each velocity pair (v, ω) in the dynamic window, simulate the resulting trajectory using the robot's motion model over a short time horizon (e.g., 1–3 seconds).

$$x_{i+1} = x_i + v \,.\, \cos(\theta_i) \,.\, \Delta t$$

$$y_{i+1} = y_i + v \,.\, \sin(\theta_i) \,.\, \Delta t$$

$$\theta_{i+1} = \theta_i + \omega \,.\, \Delta t$$

Each simulated trajectory represents a potential future path.

Step 4: Collision Check and Braking Test

- Check each simulated trajectory for possible collisions with static or dynamic obstacles using sensor data or a map.
- Additionally, compute the braking distance:

$$d_{brake} = \frac{v^2}{2 \, . \, a_{max}}$$

If an obstacle lies within this braking distance, the trajectory is marked as unsafe and discarded.

Step 5: Evaluate Cost Functions

For each remaining valid trajectory, compute a score using a weighted cost function:

$$G(v, \omega) = \alpha \, . \, G_{heading} + \beta \, . \, G_{clearance} + \gamma \, . \, G_{velocity}$$

Where:

- $G_{heading}$: Alignment with goal direction
- $G_{clearance}$: Distance to nearest obstacle
- $G_{velocity}$: Preference for faster movement

Step 6: Select the Optimal Trajectory

Choose the trajectory (and its corresponding (v, ω) command) with the highest total score.

Step 7: Execute the Command

Send the selected velocity command to the robot's actuators:

$$v_{cmd}, \omega_{cmd}$$

Step 8: Repeat

As the robot moves, the above process is repeated at every control cycle (typically 5–20 Hz), constantly adapting to changes in the environment.

7. ROS and Gazebo Simulation

The implementation and testing of the Dynamic Window Approach (DWA) in a safe and controlled environment are essential before deploying it on a real robot. This is typically

achieved using ROS (Robot Operating System) in conjunction with Gazebo, a 3D physics-based simulation tool that accurately models real-world dynamics, sensors, and environments.

7.1 Gazebo Simulator

Gazebo provides a virtual environment where robots can interact with simulated physics, including gravity, friction, and collisions. It supports 3D visualization and integrates seamlessly with ROS, making it an ideal platform for developing and debugging robotic applications. Users can create custom worlds, introduce obstacles, simulate sensor data (e.g., LiDAR, cameras), and visualize the robot's path planning behaviour.

7.2 Integration with ROS

ROS serves as the middleware that connects all the functional components of the robot, such as sensors, actuators, planners, and controllers. ROS provides standardized messages and services that allow developers to easily implement and test the DWA algorithm.

The following ROS components are commonly used in DWA-based simulations:

- `dwa_local_planner` **Package**
  This is a standard ROS package that implements the Dynamic Window Approach. It integrates with the `move_base` node in the ROS navigation stack and allows configuration of trajectory scoring, robot footprint, acceleration limits, and goal tolerance.
- `TurtleBot3` **Platform**
  TurtleBot3 is a widely used ROS-compatible differential-drive robot model. It is often chosen for simulations due to its ease of use, support from the community, and pre-built Gazebo worlds. The TurtleBot3 comes with predefined URDF models, sensor configurations, and launch files for running simulations with DWA.
- **SLAM or Map Server**
  Simulations can utilize either SLAM (Simultaneous Localization and Mapping) algorithms such as GMapping or pre-built maps using a map server. This global map is essential for global path planning, allowing DWA to work as the local planner within the broader navigation framework.
- **LiDAR Sensor**
  A simulated LiDAR sensor is used for detecting nearby obstacles. The data from LiDAR feeds into the local costmap, allowing the DWA planner to avoid collisions in real-time. ROS topics publish the laser scan data for processing by the `move_base` and `dwa_local_planner` nodes.

7.3 Configuration and Launch

The simulation is typically managed using ROS **launch files**, which start all necessary nodes and load configurations. This includes:

- Robot description files (URDF/Xacro)
- Sensor drivers
- SLAM or localization nodes
- `move_base` with `dwa_local_planner` as the local planner

- Costmap configuration (inflation radius, obstacle layers)
- Parameter tuning (e.g., DWA scoring weights: α, β, γ)

These configurations allow the user to test various scenarios, such as navigating around dynamic obstacles, adjusting planner parameters, and analyzing performance in different environments.

8. Costmap in ROS Navigation

8.1 What is a Costmap?

A costmap is a 2D grid representation of the robot's environment where each cell holds a cost value that indicates how safe or risky it is for the robot to traverse that cell.

There are two main types of costmaps in ROS:

- **Global Costmap**: Built using the static global map. Used for long-term, high-level path planning from the robot's current position to the goal.
- **Local Costmap**: Continuously updated using real-time sensor data (e.g., LiDAR, sonar). It focuses on the robot's immediate surroundings and is used for local obstacle avoidance.

8.2 How the Costmap Works

- The costmap assigns each cell a value from 0 to 255, where:
  - 0 = free space (safe to travel)
  - 255 = lethal obstacle (collision if entered)
  - ~100–254 = inflation zone (close to obstacle; avoid if possible)
- Inflation Layer: This layer "inflates" obstacle cells by assigning high costs to nearby areas, creating buffer zones. This keeps the robot from getting too close to obstacles.
- Obstacle Layer: Populated by real-time sensor input (like LiDAR scans). If an object appears in front of the robot, the corresponding grid cell becomes an obstacle.

8.3 Costmap Example

Suppose a robot sees an obstacle 1 meter ahead. The costmap will:

- Mark the obstacle cell with cost 255
- Mark surrounding cells (within, say, 0.5 m radius) with decreasing costs (254, 200, 150, etc.)
- Encourage the planner to pick trajectories that avoid these high-cost areas