Designing a Search Algorithm for Optimized PWM Selection in EVs

Abstract

This report presents the development of a search algorithm designed to optimize the selection of Pulse Width Modulation (PWM) techniques for electric vehicles (EVs). The algorithm targets the dual objectives of minimizing inverter losses (measured in watts) and torque pulsation (in percentage) while ensuring the desired speed is maintained. This document outlines the problem statement, methodology, algorithm design, implementation, and results, offering a detailed assessment of its performance and potential applications.

Introduction

1.1 Problem Context

Efficient operation of EVs relies heavily on selecting the most suitable PWM technique to control the inverter's performance and torque output. Traditional selection methods are often static and fail to dynamically account for varying operating conditions like speed and load, resulting in suboptimal efficiency.

1.2 Objectives

The primary objectives of this project are:

- To develop a search algorithm for dynamic PWM selection.
- To minimize inverter losses and torque pulsation.
- To ensure the EV achieves the desired speed with optimal performance metrics.

Methodology

2.1 Problem Analysis

The selection of PWM techniques influences the EV's performance, especially in terms of energy efficiency and mechanical stability. Critical factors include:

- **Inverter losses**: Energy lost during power conversion.
- **Torque pulsation**: Mechanical vibrations affecting ride quality and component durability.

2.2 Algorithm Approach

The algorithm evaluates the following:

- **Input Parameters**: Desired speed and constraints on inverter efficiency and torque pulsation.
- **Search Space**: A predefined set of PWM techniques.
- **Evaluation Criteria**: A weighted scoring system to balance losses and pulsation.

2.3 Maths Behind Algorithm

Tchebycheff Norm

- The **Tchebycheff norm** focuses on minimizing the **worst-case deviation** from the ideal point. The idea is to minimize the **maximum normalized deviation** from the ideal values for both objectives (inverter loss and torque pulse).

Example:

- Ideal point: `(40, 1)` for `(inverter loss, torque pulse)`.
- Technique `(40, 2)` has:
    - Deviation in inverter loss: `|40 - 40| = 0`
    - Deviation in torque pulse: `|2 - 1| = 1`
- The **Tchebycheff norm** is the maximum of the weighted deviations (in this case, 1).

Compromise Programming

- **Compromise programming** minimizes the overall distance from the ideal point, considering the **combined deviation** from the ideal values. You typically use the **p-norm** distance (e.g., Manhattan distance for p=1, Euclidean distance for p=2) to calculate the distance..

Example:

- Ideal point: `(40, 1)` for `(inverter loss, torque pulse)`.
- Technique `(40, 2)` has:
    - Deviation in inverter loss: `|40 - 40| = 0`
    - Deviation in torque pulse: `|2 - 1| = 1`
- The **Manhattan distance (p=1)** would be the sum of the deviations: `0 + 1 = 1`.

Algorithm Design

3.1 Algorithm Overview

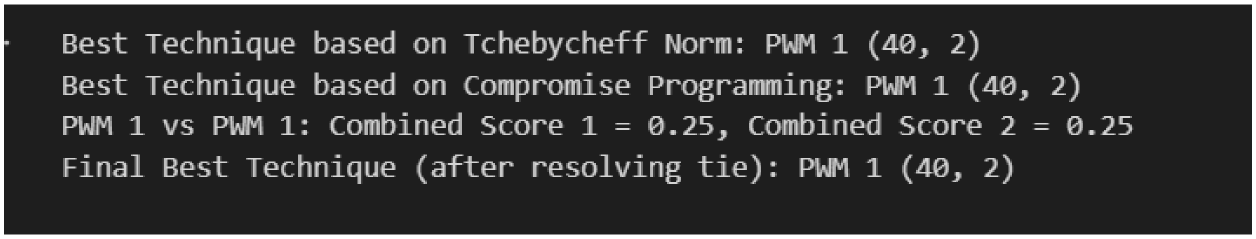
The algorithm adopts a multi-criteria decision-making framework:

1. Accepts input parameters such as desired speed and operational constraints.
2. Iteratively evaluates all available PWM techniques using the tchebycheff norm and compromise programming
3. Ranks techniques based on their performance metrics.
4. Outputs the optimal PWM technique

3.2 Code

```
import numpy as np

# Sample techniques with labels: (inverter loss, torque pulse)
techniques = [
    ("PWM 1", 40, 2),
    ("PWM 2", 50, 1),
    ("PWM 3", 60, 2),
    ("PWM 4", 45, 3)
]

# Find the ideal point based on the lowest inverter loss and tor
def find_ideal_point(techniques):
    min_inverter_loss = min(technique[1] for technique in techni
    min_torque_pulse = min(technique[2] for technique in technic
    return (min_inverter_loss, min_torque_pulse)

# Calculate ranges for normalization
def calculate_ranges(techniques):
    range_inverter_loss = max(technique[1] for technique in tech
    range_torque_pulse = max(technique[2] for technique in techn
    return (range_inverter_loss, range_torque_pulse)

# Normalize the data based on the ideal point and ranges
def normalize_data(technique, ideal_point, ranges):
    normalized = [
        (technique[1] - ideal_point[0]) / ranges[0],  # Normaliz
        (technique[2] - ideal_point[1]) / ranges[1]   # Normaliz
    ]
    return normalized

# Calculate the Tchebycheff Norm for a given technique
def calculate_tchebycheff_norm(technique, ideal_point, ranges, w
    normalized_deviations = normalize_data(technique, ideal_poin
    tchebycheff_norm = max(weight1 * abs(normalized_deviations[0
    return tchebycheff_norm

# Calculate the Compromise Distance (Manhattan or Euclidean)
def calculate_compromise_distance(technique, ideal_point, ranges
    normalized_deviations = normalize_data(technique, ideal_poin
    if p == 1:
        distance = weight1 * abs(normalized_deviations[0]) + wei
    elif p == 2:
        distance = (weight1 * (normalized_deviations[0])**2 + we
    return distance

# Resolve a tie by combining the results from both methods with
def resolve_tie_with_weighting(technique1, technique2, ideal_poi
    tchebycheff1 = calculate_tchebycheff_norm(technique1, ideal_
    tchebycheff2 = calculate_tchebycheff_norm(technique2, ideal_

    compromise1 = calculate_compromise_distance(technique1, idea
    compromise2 = calculate_compromise_distance(technique2, idea

    # Calculate combined scores
    combined_score1 = weight_tchebycheff * tchebycheff1 + weight
    combined_score2 = weight_tchebycheff * tchebycheff2 + weight

    print(f"{technique1[0]} vs {technique2[0]}: Combined Score 1

    return technique1 if combined_score1 < combined_score2 else

# Evaluate all techniques based on both methods
def evaluate_techniques(techniques):
    ideal_point = find_ideal_point(techniques)
    ranges = calculate_ranges(techniques)

    best_technique_tchebycheff = techniques[0]
    best_technique_compromise = techniques[0]

    for technique in techniques[1:]:
        # Evaluate Tchebycheff Norm method
        if calculate_tchebycheff_norm(technique, ideal_point, ra
            best_technique_tchebycheff = technique

        # Evaluate Compromise Programming method
        if calculate_compromise_distance(technique, ideal_point,
            best_technique_compromise = technique
    print(f"Best Technique based on Tchebycheff Norm: {best_tech
    print(f"Best Technique based on Compromise Programming: {bes

    # Resolve the tie if needed and print the final technique
    final_best_technique = resolve_tie_with_weighting(best_techn
    print(f"Final Best Technique (after resolving tie): {final_b

# Example usage:
evaluate_techniques(techniques)
```

Result



```
Best Technique based on Tchebycheff Norm: PWM 1 (40, 2)
Best Technique based on Compromise Programming: PWM 1 (40, 2)
PWM 1 vs PWM 1: Combined Score 1 = 0.25, Combined Score 2 = 0.25
Final Best Technique (after resolving tie): PWM 1 (40, 2)
```

Fig 4.1 Output