

Report on
Designing a Search Algorithm for Optimized PWM
Selection in EVs

Dhinakar.S.P

B.Tech CSE (Cyber Security) – Shiv Nadar University, Chennai

Submitted to:

Mr. Kathirvel

Hardware Team Manager - Power Electronics

Valeo India Pvt Ltd.



Table of Contents

S.No	Name	Page No.
1.	Abstract	3
2.	Methodology	3
3.	Algorithm Design	5
4.	Result	9
5.	Limitation	9
6.	Prospect	10
7.	Conclusion	10

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.

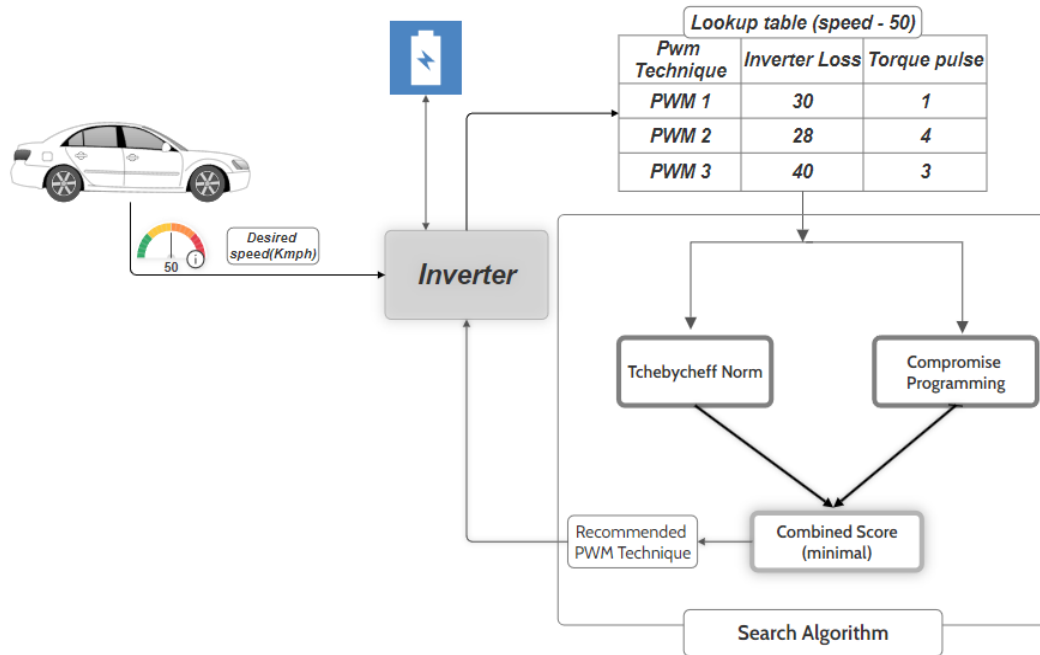


Fig.2.2.1 Working of Algorithm

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 torque pulse
def find_ideal_point(techniques):
    min_inverter_loss = min(technique[1] for technique in techniques)
    min_torque_pulse = min(technique[2] for technique in techniques)
    return (min_inverter_loss, min_torque_pulse)

# Calculate ranges for normalization
def calculate_ranges(techniques):
    range_inverter_loss = max(technique[1] for technique in techniques) -
min(technique[1] for technique in techniques)
    range_torque_pulse = max(technique[2] for technique in techniques) -
min(technique[2] for technique in techniques)
    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], # Normalized inverter loss
        (technique[2] - ideal_point[1]) / ranges[1] # Normalized torque pulse
    ]
    return normalized

# Calculate the Tchebycheff Norm for a given technique
def calculate_tchebycheff_norm(technique, ideal_point, ranges, weight1=0.5,
weight2=0.5):

```

```

    normalized_deviations = normalize_data(technique, ideal_point, ranges)

    tchebyscheff_norm = max(weight1 * abs(normalized_deviations[0]), weight2 *
abs(normalized_deviations[1]))

    return tchebyscheff_norm

# Calculate the Compromise Distance (Manhattan or Euclidean)

def calculate_compromise_distance(technique, ideal_point, ranges, weight1=0.5,
weight2=0.5, p=1):

    normalized_deviations = normalize_data(technique, ideal_point, ranges)

    if p == 1:

        distance = weight1 * abs(normalized_deviations[0]) + weight2 *
abs(normalized_deviations[1])

    elif p == 2:

        distance = (weight1 * (normalized_deviations[0])**2 + weight2 *
(normalized_deviations[1])**2)**0.5

    return distance

# Resolve a tie by combining the results from both methods with weights

def resolve_tie_with_weighting(technique1, technique2, ideal_point, ranges,
weight_tchebyscheff=0.5, weight_compromise=0.5):

    tchebyscheff1 = calculate_tchebyscheff_norm(technique1, ideal_point, ranges)

    tchebyscheff2 = calculate_tchebyscheff_norm(technique2, ideal_point, ranges)

    compromise1 = calculate_compromise_distance(technique1, ideal_point, ranges)

    compromise2 = calculate_compromise_distance(technique2, ideal_point, ranges)

# Calculate combined scores

    combined_score1 = weight_tchebyscheff * tchebyscheff1 + weight_compromise *
compromise1

```

```

    combined_score2 = weight_tchebycheff * tchebycheff2 + weight_compromise *
    compromise2

    return technique1 if combined_score1 < combined_score2 else technique2

# 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, ranges) <
calculate_tchebycheff_norm(best_technique_tchebycheff, ideal_point, ranges):

            best_technique_tchebycheff = technique

        # Evaluate Compromise Programming method

        if calculate_compromise_distance(technique, ideal_point, ranges) <
calculate_compromise_distance(best_technique_compromise, ideal_point, ranges):

            best_technique_compromise = technique

    # Compare and handle the case of a tie

    if best_technique_tchebycheff == best_technique_compromise:

        print(f"Recommended Technique: {best_technique_tchebycheff[0]}
({best_technique_tchebycheff[1]}, {best_technique_tchebycheff[2]})")

    else:

```



```


final_best_technique = resolve_tie_with_weighting(
    best_technique_tchebycheff,
    best_technique_compromise,
    ideal_point,
    ranges
)

print(f"Recommended Technique: {final_best_technique[0]}
({final_best_technique[1]}, {final_best_technique[2]})")

# Example usage:
evaluate_techniques(techniques)

```

Result



```

[3] ✓ 0.0s Python
... Recommended Technique: PWM 1 (40, 2)

```

Fig 4.1 Sample Output

The recommended technique is the output for the desired speed in EVs. Here if there is any tie in the combined score even after the mathematical considerations, the first available technique is chosen as the recommended technique.

Limitation

- The parameters involved are certainly not enough to give a optimal PWM technique.
- After all the mathematical calculations, there is a mere chance for a tie to arise in the techniques, this may bring the first available technique to chose but may not be the optimal sometimes.

Prospects

- Increase in the no. of parameters involved.
- Assign weightage for parameters based on the EVs to break the tie.

Conclusion

This report presents a search algorithm for optimizing Pulse Width Modulation (PWM) techniques in electric vehicles (EVs), aiming to minimize inverter losses and torque pulsation while having the desired speed. The algorithm uses multi-criteria decision-making methods, including the Tchebycheff norm and compromise programming, to evaluate and rank PWM techniques based on performance metrics. Though the algorithm effectively identifies the best technique, it faces limitations such as potential ties between techniques when scores are close. Future improvements could involve incorporating additional parameters and adjusting weightings based on specific EV characteristics for more refined decisions. Overall, the algorithm enhances the efficiency and stability of EV operation.