Engineering Challenge 1 - Weight Reduction vs Ride Height on a Mazda MX-5 at Charlotte Roval

Sussex Racing, Lucas Albery, Dan Hawkins

Date: Friday 5^{th} November 2024

Abstract

This report optimises a Mazda MX-5 Cup Car on Charlotte Roval by balancing weight reduction and ride height adjustments within a £25,000 budget. MATLAB simulations and experimental trials were conducted to evaluate their impact on lap times. Results reveal that while weight reduction directly improves acceleration and handling, excessive reductions can destabilize the car. Ride height adjustments enhance aerodynamics and stability, with a balanced configuration yielding optimal performance. The best setup was a 10 kg weight reduction with a 10-click front and 2-click rear ride height adjustment, achieving the fastest lap time of 1:34.8.

Nomenclature

μ : Tire Friction Coefficient

C_d : Drag Coefficient

C_l : Lift Coefficient

kg : Kilogram

$MX-5$: Mazda MX-5 Cup Car

Contents

1. Introduction

This report is about performance of a Mazda MX-5 Cup Car on the Charlotte Roval track by balancing

two parameters: weight reduction and ride height adjustment. The study aims to determine the best compromise to achieve the fastest lap times within a £25,000 budget.

1.1. Objectives

(i) Outline how car performance changes when weight is reduced from the car and the ride height is adjusted.
(ii) Determine an ideal compromise between the two parameters without exceeding the budget
(iii) Recommend a setup parameter to commit to and justify it.
(iv) Use the old car to post your fastest lap on the server to determine your benchmark.

1.2. Paramaters

(i) Track = Charlotte Roval
(ii) Car = Mazda Mx5 Cup Car
(iii) Tyre Wear will be switched off
(iv) Brake Bias is Open
(v) Max Budget to spend = £25,000
(vi) Ride height adjustment costs £625 per click from the original setting (per corner)
(vii) Weight reduction = £1000 per kg reduced.

2. Possible configurations to test

Possible combinations using the maximum budget are shown using the matlab script shown in the appendix. It revealed the graph shown in Fig. 1.

This gives us six possible configurations. This means we will be able to figure out the correct configuration out of three six using experimental methods.

3. Hypothesis

The optimal configuration for maximising car performance within the given budget constraints will involve prioritising a combination of ride height adjustment and weight reduction, favoring weight reduction more heavily.

The assumption is that reducing the car's weight will directly enhance acceleration, braking, and cornering performance, which are critical factors for lap times.

Although, lowering the ride height can improve aerodynamics and stability by reducing drag and increasing downforce, the incremental costs associated with lowering each corner make it less cost-effective compared to weight reduction in achieving significant performance gains.

Among the six configurations that utilise the maximum budget, we currently believe the most balanced configuration will involve moderate ride height reduction paired with a significant weight reduction, as this should yield the greatest net improvement in lap times while staying within financial constraints.

Our hypothesis will be validated by testing these configurations on the Charlotte Roval track and analysing their impacts on lap times relative to the benchmark set with the old car.

4. Calculations

4.1. Effect of Weight Reduction

To calculate the effect of weight reduction on lap times for the Mazda MX-5 Cup Car around the Charlotte Roval track, each kilogram of weight reduction costs £1,000, so the maximum weight reduction possible within a £25,000 budget is 25 kg.

The Charlotte Roval Track is shown in Fig. 2. As can be seen, there are 17 turns.

4.1.1. MATLAB script

We will write a MATLAB script that:
(i) Calculates the maximum achievable weight reduction. (ii) Estimates lap time improvements based on different weight reductions. (iii) Plots a graph showing the relationship between weight reduction and lap time.

MATLAB script can be found in the appendix. Results however, can be found here, in Fig. 3.

For this MATLAB code parameters needed to be considered. Parameters found are in the following subsubsections.

Downforce Coefficient

For the Mazda MX-5, the downforce coefficient C_d is not extensively documented in public literature due to its focus on road performance rather than high-downforce aerodynamics. However, rough estimates based on similar lightweight sports cars suggest a lift coefficient (negative downforce) around -0.2 to -0.3 for stock configurations. (Car and Driver, 2023)

Tyre Friction Coefficient

For the Mazda MX-5 equipped with stock high-performance tires like the Bridgestone Potenza S001: - Dry Surface: Coefficient of friction ranges from approximately 0.9 to 1.1.(Bridgestone Corporation, 2023) - Wet Surface: Coefficient of friction is lower, around 0.6 to 0.8, depending on road conditions and tire wear. (Bridgestone Corporation, 2023)

We then defined our track by defining straight lengths, corner lengths, corner radius' and total distance. This allowed for two seperate sections of calculation: straight and corners.

In the straights there were also two bits of calculation: that of when top speed was achieved, i.e. cruising, and when it was not and the car was still cruising. This allowed us to figure out how long the car took in the straights. We simply added that to the time the car took in the corners with the corner equations. We then ran a for loop doing that for

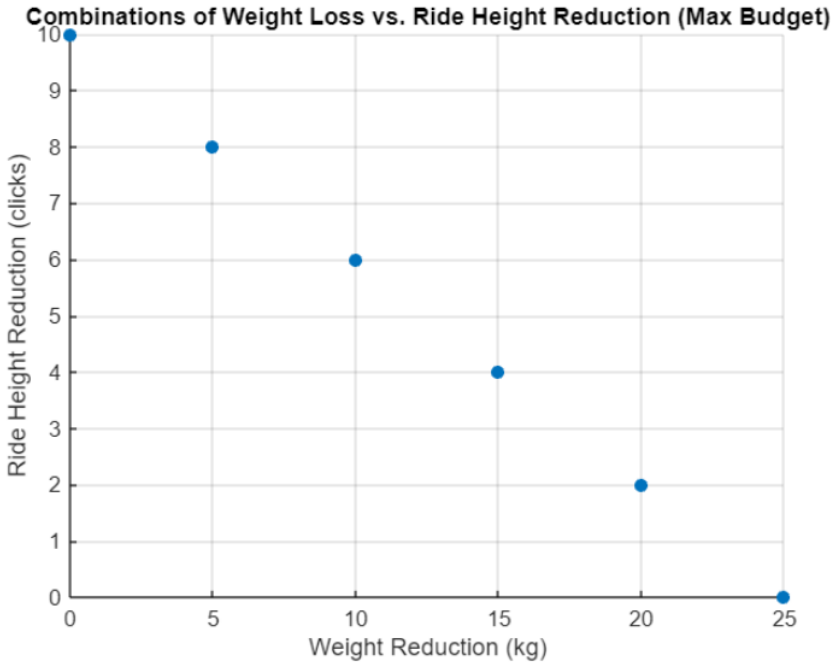


Fig. 1: Possible combinations using the maximum budget

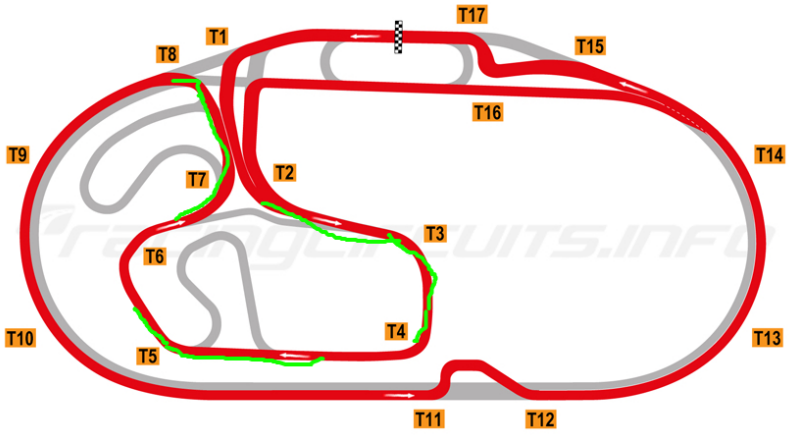


Fig. 2: Charlotte Roval

weight reductions of 1 - 25 kg.

4.2. Discussion of Weight Reduction Results

For the MATLAB code, the code assumes that there is a perfect driver, but in fact so perfect that they are never in the wrong place on the track. The separation of corner and straight sections on the track means that the driver is in the best possible place for all parts of the corner, but then immediately is in the best place for the straight because the track does not really exist. This means that the car in any theoretical track constructed would teleport.

However, it does give an idea about how weight would affect lap times. Lap times generally linearly reduce with weight reduction and a perfect driver, as seen in Fig, 3.

4.3. Effect of Ride Height Adjustment

To calculate the effect of ride height adjustment on lap times for the Mazda MX-5 Cup Car around

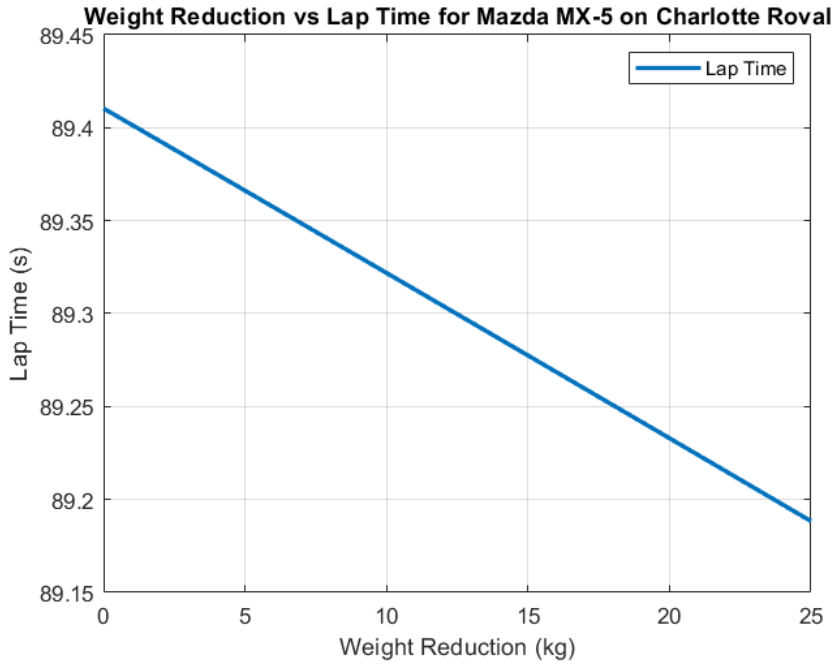


Fig. 3: Weight Reduction vs Lap Time for Ma zda MX-5 on Charlotte Roval

the Charlotte Roval track, each click of ride height adjustment costs £625 per corner. Therefore, the maximum achievable ride height reduction within the £25,000 budget is $\frac{25{,}000}{4\times625} = 10$ clicks per corner.

4.3.1. MATLAB Script

For this our MATLAB script will:

(i) Calculates the maximum achievable ride height adjustment.

(ii) Estimates lap time improvements based on different ride height adjustments.

(iii) Plots a graph showing the relationship between ride height adjustment and lap time.

The MATLAB script can be found in the appendix. Results, however, are summarized here in Fig. 4.

For this MATLAB code, parameters needed to be considered. Parameters used are described in the following subsubsections.

Aerodynamic Downforce and Drag Coefficients

For the Mazda MX-5, ride height adjustments influence both aerodynamic drag (C_d) and lift/downforce coefficients (C_l). A lower ride height generally reduces drag and increases downforce, but diminishing returns occur beyond certain points due to ground effects. Approximate coefficients for ride height reductions are as follows (derived from generic sports car data and adjusted for the MX-5):

- Default ride height: $C_d = 0.38$, $C_l = -0.25$
- 5-click reduction: $C_d = 0.36$, $C_l = -0.28$
- 10-click reduction: $C_d = 0.34$, $C_l = -0.30$

Tire Contact Patch and Friction Coefficient

Lowering the ride height reduces body roll and improves tire contact patch utilization in corners, increasing the effective tire friction coefficient. The coefficient of friction increases slightly with ride height reductions, modeled approximately as:

$$\mu = \mu_0 + k \cdot \text{clicks\_reduced}$$

Where $\mu_0 = 1.0$ (baseline tire friction coefficient), and $k = 0.01$ (increment per click).

Track Definition and Ride Height Impact

The track was defined similarly to the weight reduction analysis by separating straight and corner sections. For ride height, we assumed the following:

- **Straights**: Lower ride height marginally improves drag coefficient, increasing top speed.
- **Corners**: Improved downforce and tire friction reduce cornering times.

A MATLAB script calculated lap times for different ride height reductions by iteratively adjusting aerodynamic and friction parameters for each click reduction, while separating straights and corners for detailed calculations.

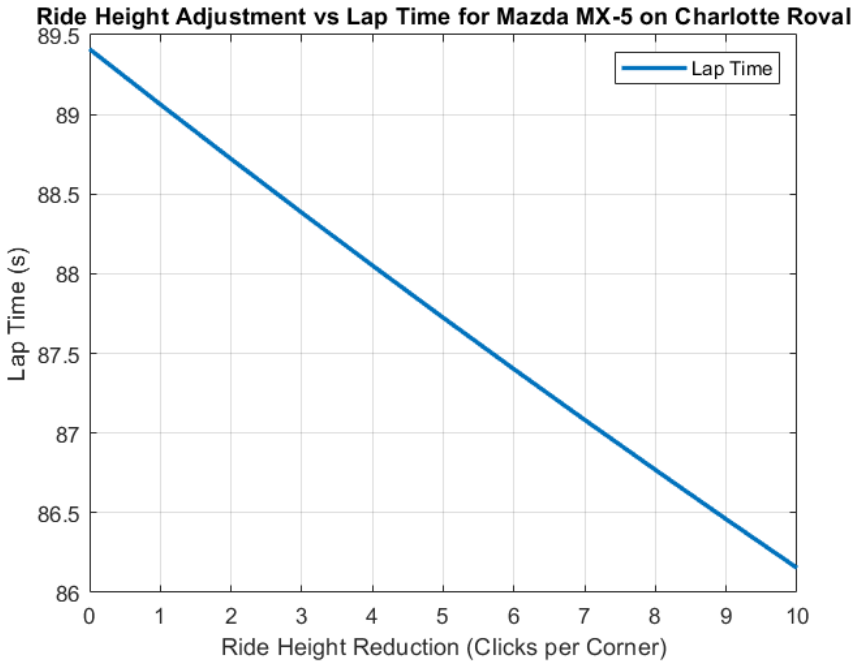


Fig. 4: Ride Height Adjustment vs Lap Time for Mazda MX-5 on Charlotte Roval

4.3.2. Discussion of Ride Height Adjustment Results

The MATLAB analysis assumes a perfect driver who utilizes the car's theoretical maximum performance in all scenarios, including the aerodynamic benefits of reduced drag and downforce improvements. As with weight reduction, the track is idealized, meaning there are no imperfections in surface or variability in driving style.

The results in Fig. 4 show that lap times decrease with ride height reductions due to improved aerodynamics and cornering performance.

4.4. Comparison of the two

The idea of comparing the two is to find the best setup using both elements.

4.4.1. MATLAB code

For this a combined MATLAB code was created which is also in the appendix.

The code analyses weight and ride height adjustments on lap time performance within a given budget constraint. It evaluates individual and combined effects of these adjustments and visualizes their influence on lap times.

It defines vehicle properties (mass, power, aerodynamics, and tire properties) then track details and then specifies costs and budget constraints for adjustments.

Weight reduction calculates lap times for various weight reductions by lowering vehicle mass.

Ride height adjustment adjusts drag coefficient and tire friction incrementally, computing their impact on lap times.

Reversed ride height lap times reverses the lap_times_height array for direct comparison with weight reduction trends.

The script then goes through combinations of weight and ride height sticking to budget. It then stores lap times and budgets for feasible combinations and then computes average lap times for unique budget points.

As a function the old scripts are integrated into the new script. In the cornering it uses turn radius and tire friction to compute time spent in corners. The straight section models acceleration and cruising phases to determine time spent on straights.

It outputs the graph you can see in Fig. 5.

The results in this figure do not make sense, especially when compared to the experimental data section. The results say that the best thing to do is to decrease the ride height only. From the experimental section, we believe that the results should be a lot more balanced.

However, we can see that the problem is the weight reduction. During debugging, we could see that even if we put the weight down to 1 the lap time would only be 70 seconds. Which should not happen with the huge power to weight ratio.

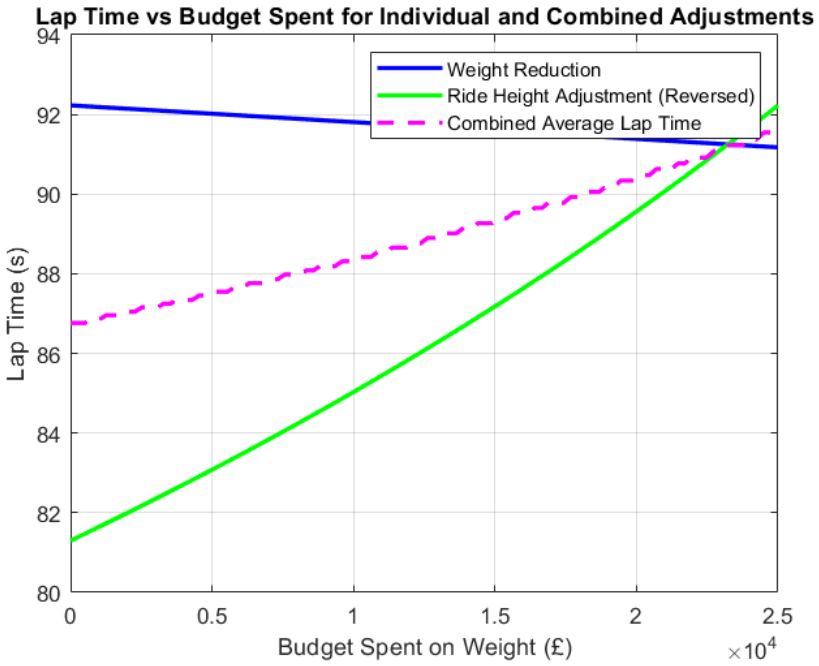


Fig. 5: Comparison of weight to ride height

5. Experimental Data

5.1. Lap Times

The results in table 1, it shows the times our engineer driver, Dan Hawkins, achieved in Asseto Corsa with different set ups.

In Fig. 6, it shows the times on a graph.

The fastest times achieved was by dropping the time by 10kg and by doing 10 clicks on the front and 2 clicks on the rear. This is similar to the driver notes.

5.2. Driver Notes

I found I think a 50/50 split of weight reduction and ride height feels nicest, I prefer having rear wheel height lowered over front for corner exits but that's just preference.

5.3. Discussion

5.3.1. Weight Reduction

Reducing vehicle weight generally improves performance by lowering the overall inertia, allowing quicker acceleration and shorter braking distances.

However, there's a balance to be struck. -10kg Reduction: The best lap time (1:34.8) was achieved here, likely because it reduced weight sufficiently to enhance responsiveness without overly destabilizing the car.

5.3.2. Suspension Tuning

Suspension settings directly affect grip, handling balance, and the car's response to driver inputs.

10 Front / 2 Rear Setup: This configuration likely provided a balanced approach by allowing the front suspension to absorb impacts and maintain cornering grip while keeping the rear more rigid for better power transfer during acceleration.

Uniform Adjustments: Adjustments like "10 clicks down each corner" evenly altered stiffness across the car, which may have lacked the nuanced tuning needed for optimal performance on specific track sections.

Extreme Adjustments: Setups such as "4 clicks front, 0 clicks rear" was described as undrivable because it created significant imbalances, making the car unpredictable and difficult to control.

5.3.3. Stability and Handling

Hawkins' notes about stability align with the observed data. For example:

Unstable Under Braking: When too much height or weight was removed, the car likely became prone to oversteer or instability during heavy braking zones, increasing lap times.

Corner Exit Preference: A lowered rear ride height favors traction on corner exits by shifting more weight to the rear tires, providing better grip during acceleration.

Suspension	Weight	Time
10 clicks down each suspension corner	-	1.40.805 / (redone) 1.36.608
8 clicks down each corner	-5kg	1.38.595
6 clicks down each corner	-10kg	1.37.406 (More unstable - under braking)
4 clicks down each corner	-15kg	1.38.117
2 clicks down each corner	-20kg	1.37.129
-	-25kg	1.36.403 (Understeery)
4 clicks on front suspension, 0 clicks on back suspension	-20kg	Pretty undriveable
10 clicks front, 2 rear	-10kg	1.34.8

Table 1: Suspension Setup Times

Fig. 6: Weight vs Lap Times in Sim Racing with Suspension Labels

5.3.4. Driver Preference

Setup optimization often depends on the driver's style. Hawkins' preference for a "50/50 split" and lowered rear height reflects a subjective comfort level that allows them to push the car effectively. A setup aligned with these preferences allows better consistency and confidence, which translates into faster lap times.

5.3.5. Trade-offs

Weight Reduction vs. Stability: Lighter cars are faster but harder to control, especially with the the greater ride height.

6. Conclusion

The optimal setup combines a moderate weight reduction with targeted ride height adjustments, prioritizing balance over extremes. Specifically, a configuration of a 10 kg weight reduction and a front-rear ride height adjustment of 10 clicks and 2 clicks, respectively, delivered the fastest lap while maintaining drivability. This approach aligns with the hypothesis but underscores the value of tailored tuning based on driver preferences and track characteristics.

7. Appendix

o

.

Table of Contents


```
% MATLAB Code for Weight Reduction Analysis
clear; clc;
```

Constants and Initial Data

Vehicle parameters

```
mass_original = 1100; % Vehicle mass in kg (including driver)
power = 180; % Power in kW
drag_coeff = 1; % Drag coefficient
frontal_area = 2.2; % Frontal area in m^2
tire_mu = 1; % Coefficient of friction
g = 9.81; % Gravity in m/s^2

% Track parameters
track_length = 3800; % Track length in meters
turn_radii = [30, 50, 70, 100, 60, 80, 50, 40, 90, 70, 110, 120, 30, 60, 80,
50, 100]; % Turn radii in meters
num_turns = length(turn_radii);
corner_lengths = 2 * pi * turn_radii / 360 * 90; % 90-degree corners for
simplicity
total_corner_length = sum(corner_lengths);
straight_lengths = track_length - total_corner_length; % Total straight
length divided among straights
avg_straight_length = straight_lengths / (num_turns - 1); % Average straight
length
air_density = 1.225; % Air density in kg/m^3

% Analysis parameters
weight_reductions = 0:1:25; % Weight reductions in kg
```

Pre-allocations

```
lap_times = zeros(size(weight_reductions)); % Lap times for different weights
```

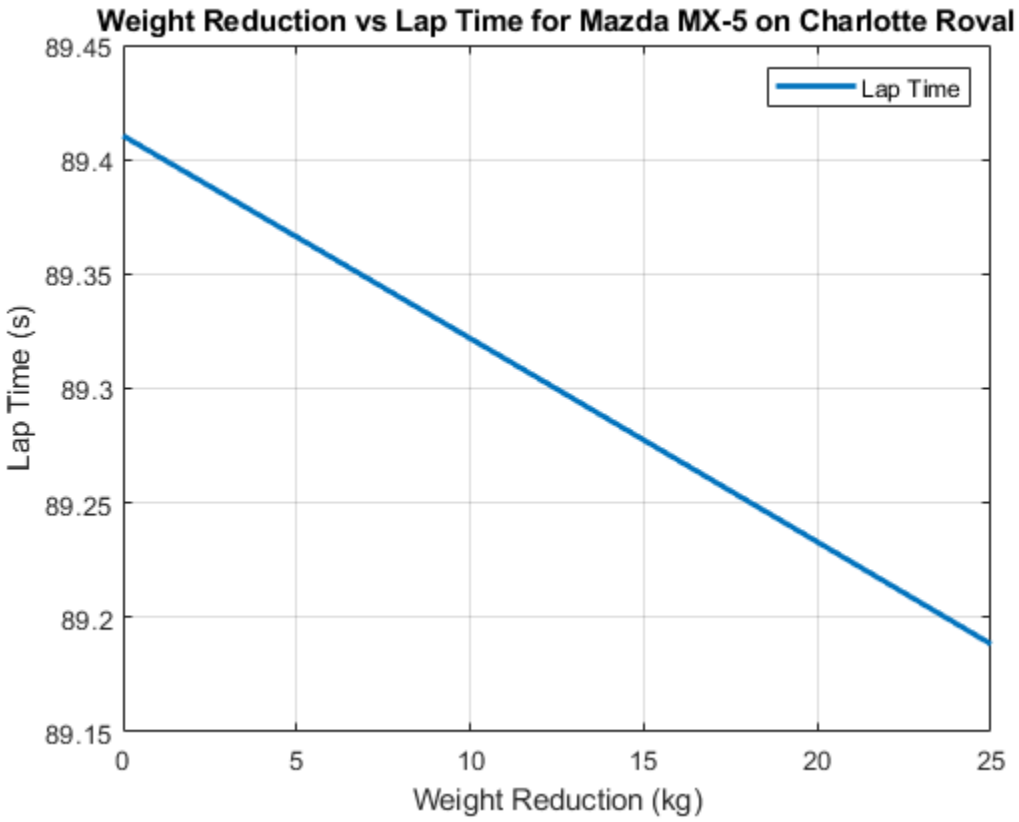
Simulation Loop

```
for i = 1:length(weight_reductions)
    mass = mass_original - weight_reductions(i); % Reduced mass
    lap_time = 0; % Reset lap time
```

```
    % Cornering
    for j = 1:num_turns
        radius = turn_radii(j);
        max_cornering_speed = sqrt(tire_mu * g * radius); % Max speed in
corner
        corner_time = corner_lengths(j) / max_cornering_speed; % Time to
complete the corner
        lap_time = lap_time + corner_time;
    end

    % Straights
    straight_accel = (power * 1000) / mass; % Acceleration on straights
    top_speed = sqrt((2 * power * 1000) / (drag_coeff * air_density *
frontal_area)); % Approximate top speed

    for j = 1:(num_turns - 1) % Number of straights = corners - 1
        % Acceleration phase
        d_accel = min((top_speed^2) / (2 * straight_accel),
avg_straight_length); % Distance covered during acceleration
        t_accel = sqrt(2 * d_accel / straight_accel); % Time to cover
acceleration phase

        % Cruising phase
        if d_accel < avg_straight_length
            d_cruise = avg_straight_length - d_accel; % Remaining distance
            t_cruise = d_cruise / top_speed; % Cruising time
        else
            t_cruise = 0; % No cruising
        end

        % Add straight time to lap time
        lap_time = lap_time + t_accel + t_cruise;
    end

    % Store lap time
    lap_times(i) = lap_time;
end
```

Plot Results

```
figure;
plot(weight_reductions, lap_times, 'LineWidth', 2);
grid on;
xlabel('Weight Reduction (kg)');
ylabel('Lap Time (s)');
title('Weight Reduction vs Lap Time for Mazda MX-5 on Charlotte Roval');
legend('Lap Time');
```



Published with MATLAB® R2024b

Table of Contents


```
% MATLAB Code for Ride Height Analysis
clear; clc;
```

Constants and Initial Data

Vehicle parameters

```
mass = 1100; % Vehicle mass in kg (including driver)
power = 180; % Power in kW
drag_coeff_base = 0.38; % Baseline drag coefficient
lift_coeff_base = -0.25; % Baseline lift/downforce coefficient
frontal_area = 2.2; % Frontal area in m^2
tire_mu_base = 1.0; % Baseline coefficient of friction
g = 9.81; % Gravity in m/s^2

% Track parameters
track_length = 3800; % Track length in meters
turn_radii = [30, 50, 70, 100, 60, 80, 50, 40, 90, 70, 110, 120, 30, 60, 80,
50, 100]; % Turn radii in meters
num_turns = length(turn_radii);
corner_lengths = 2 * pi * turn_radii / 360 * 90; % 90-degree corners for
simplicity
total_corner_length = sum(corner_lengths);
straight_lengths = track_length - total_corner_length; % Total straight
length divided among straights
avg_straight_length = straight_lengths / (num_turns - 1); % Average straight
length
air_density = 1.225; % Air density in kg/m^3

% Ride height adjustment parameters
clicks_reduced = 0:1:10; % Ride height adjustments (clicks per corner)
drag_coeff_reduction = 0.02; % Reduction in drag coefficient per click
lift_coeff_increase = 0.01; % Increase in downforce coefficient per click
tire_mu_increase = 0.01; % Increase in tire friction coefficient per click
```

Pre-allocations

```
lap_times = zeros(size(clicks_reduced)); % Lap times for different ride
height adjustments
```

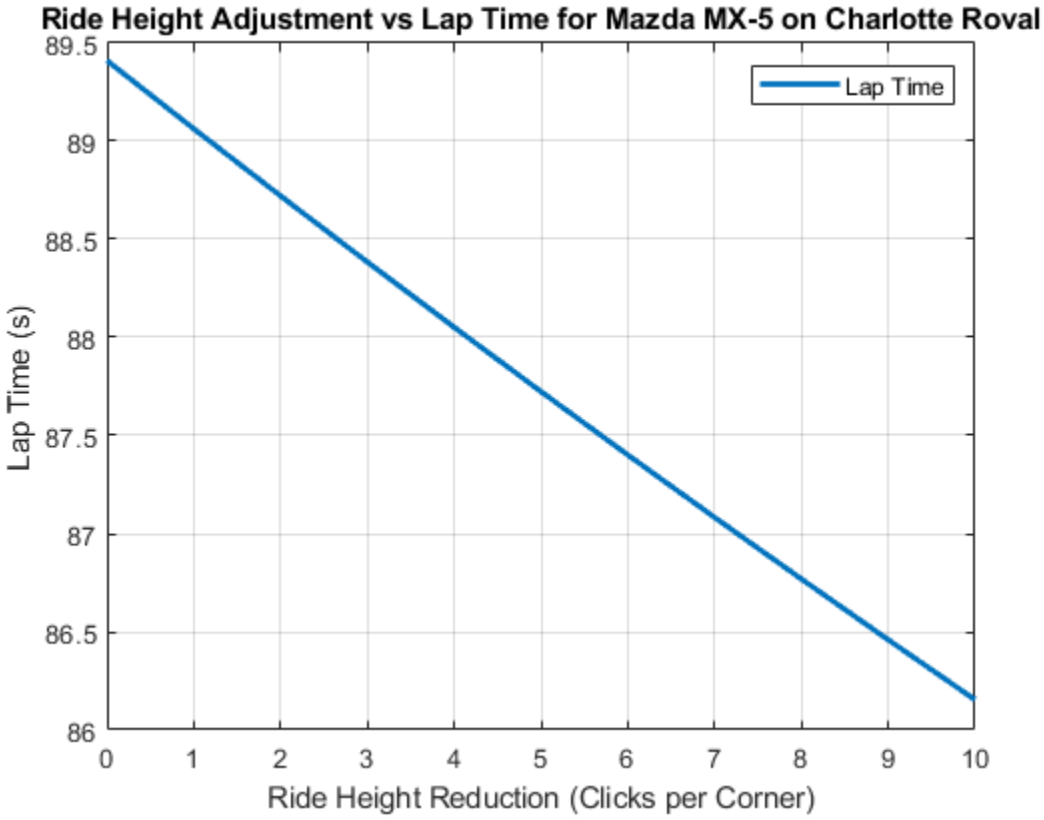
Simulation Loop

```
for i = 1:length(clicks_reduced)
    % Adjust aerodynamic and tire parameters
    drag_coeff = drag_coeff_base - clicks_reduced(i) * drag_coeff_reduction;
    lift_coeff = lift_coeff_base - clicks_reduced(i) * lift_coeff_increase;
    tire_mu = tire_mu_base + clicks_reduced(i) * tire_mu_increase;

    lap_time = 0; % Reset lap time

    % Cornering
    for j = 1:num_turns
        radius = turn_radii(j);
        max_cornering_speed = sqrt(tire_mu * g * radius); % Max speed in
corner
        corner_time = corner_lengths(j) / max_cornering_speed; % Time to
complete the corner
        lap_time = lap_time + corner_time;
    end

    % Straights
    straight_accel = (power * 1000) / mass; % Acceleration on straights
    top_speed = sqrt((2 * power * 1000) / (drag_coeff * air_density *
frontal_area)); % Approximate top speed

    for j = 1:(num_turns - 1) % Number of straights = corners - 1
        % Acceleration phase
        d_accel = min((top_speed^2) / (2 * straight_accel),
avg_straight_length); % Distance covered during acceleration
        t_accel = sqrt(2 * d_accel / straight_accel); % Time to cover
acceleration phase

        % Cruising phase
        if d_accel < avg_straight_length
            d_cruise = avg_straight_length - d_accel; % Remaining distance
            t_cruise = d_cruise / top_speed; % Cruising time
        else
            t_cruise = 0; % No cruising
        end

        % Add straight time to lap time
        lap_time = lap_time + t_accel + t_cruise;
    end

    % Store lap time
    lap_times(i) = lap_time;
end
```

Plot Results

```
figure;
plot(clicks_reduced, lap_times, 'LineWidth', 2);
```

```
grid on;
xlabel('Ride Height Reduction (Clicks per Corner)');
ylabel('Lap Time (s)');
title('Ride Height Adjustment vs Lap Time for Mazda MX-5 on Charlotte
Roval');
legend('Lap Time');
```



Published with MATLAB® R2024b

Table of Contents


```
% MATLAB Code for Weight and Ride Height Analysis with Enhanced Performance
Impact
clear; clc;
```

Constants and Initial Data

Vehicle parameters

```
mass_original = 1100; % Vehicle mass in kg (including driver)
power = 145; % Power in kW
drag_coeff_base = 1; % Baseline drag coefficient
tire_mu_base = 1; % Baseline coefficient of friction
frontal_area = 2.2; % Frontal area in m^2
g = 9.81; % Gravity in m/s^2
air_density = 1.225; % Air density in kg/m^3

% Track parameters
track_length = 3800; % Track length in meters
turn_radii = [30, 50, 70, 100, 60, 80, 50, 40, 90, 70, 110, 120, 30, 60, 80,
50, 100]; % Turn radii in meters
num_turns = length(turn_radii);
corner_lengths = 2 * pi * turn_radii / 360 * 90; % 90-degree corners for
simplicity
total_corner_length = sum(corner_lengths);
straight_lengths = track_length - total_corner_length; % Total straight
length divided among straights
avg_straight_length = straight_lengths / (num_turns - 1); % Average straight
length

% Adjustment parameters and costs
cost_per_kg_weight = 1000; % Cost per kg of weight reduction
cost_per_click = 625; % Cost per click of ride height adjustment
budget_limit = 25000; % Total budget in $

% Maximum reductions based on budget
max_weight_reduction = floor(budget_limit / cost_per_kg_weight);
max_clicks = floor(budget_limit / cost_per_click);
```

Lap Time Calculation for Individual Parameters

Arrays for weight and ride height analyses

```
budget_weight = (0:max_weight_reduction) * cost_per_kg_weight;
lap_times_weight = zeros(size(budget_weight));

budget_height = (0:max_clicks) * cost_per_click;
lap_times_height = zeros(size(budget_height));

% Calculate lap times for weight reduction
for i = 1:length(budget_weight)
    weight_reduction = i - 1;
    mass = mass_original - weight_reduction; % Decrease mass
    lap_times_weight(i) = calculate_lap_time(mass, drag_coeff_base,
tire_mu_base, power, num_turns, corner_lengths, avg_straight_length, g,
air_density, frontal_area, mass_original);
end

% Calculate lap times for ride height adjustment
for i = 1:length(budget_height)
    clicks_reduced = i - 1;
    drag_coeff = drag_coeff_base - clicks_reduced * 0.02; % Reduction per
click
    tire_mu = tire_mu_base + clicks_reduced * 0.01; % Increase per click
    lap_times_height(i) = calculate_lap_time(mass_original, drag_coeff,
tire_mu, power, num_turns, corner_lengths, avg_straight_length, g,
air_density, frontal_area, mass_original);
end
```

Reverse the lap_times_height array using a loop

```
reversed_lap_times_height = zeros(size(lap_times_height));
for i = 1:length(lap_times_height)
    reversed_lap_times_height(i) = lap_times_height(end - i + 1);
end
```

Combined Analysis for All Combinations

Initialize arrays to store results

```
budget_points = [];
lap_times = [];
average_lap_times = [];

% Iterate through all combinations of weight reduction and ride height clicks
for weight_reduction = 0:max_weight_reduction
    for clicks_reduced = 0:max_clicks
        % Calculate the total budget spent for this combination
        budget_spent = weight_reduction * cost_per_kg_weight +
```

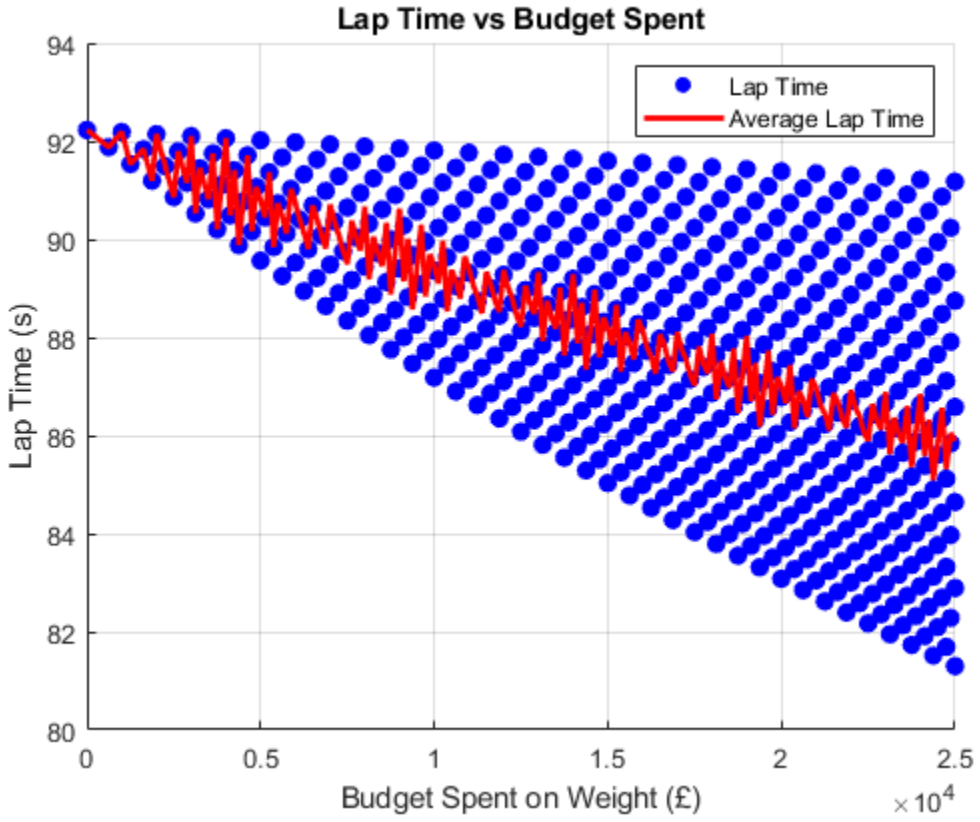
```
clicks_reduced * cost_per_click;

        % Check if the budget is within the limit
        if budget_spent <= budget_limit
            % Adjust vehicle parameters
            mass = mass_original - weight_reduction;
            drag_coeff = drag_coeff_base - clicks_reduced * 0.02; %
Reduction per click
            tire_mu = tire_mu_base + clicks_reduced * 0.01; % Increase per
click

            % Calculate lap time for this combination
            lap_time = calculate_lap_time(mass, drag_coeff, tire_mu, power,
...
                num_turns, corner_lengths, avg_straight_length, g,
air_density, frontal_area, mass_original);

            % Store results
            budget_points = [budget_points; budget_spent];
            lap_times = [lap_times; lap_time];
        end
    end
end

% Calculate the average lap time at each unique budget point
[unique_budget, ~, idx] = unique(budget_points);
average_lap_times = accumarray(idx, lap_times, [], @mean);
```

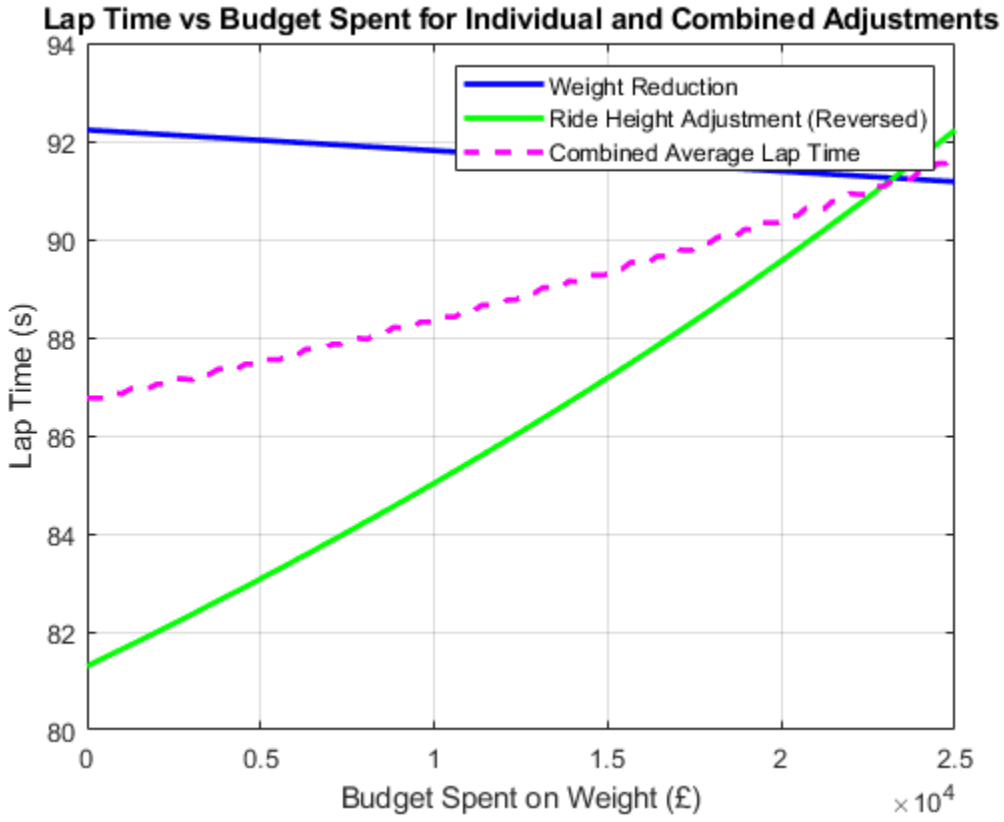
Plot Results

Plot individual lap times and average lap time

```
figure;
scatter(budget_points, lap_times, 50, 'b', 'filled', 'DisplayName', 'Lap
Time'); % Scatter plot of individual lap times
hold on;
plot(unique_budget, average_lap_times, 'r-', 'LineWidth', 2, 'DisplayName',
'Average Lap Time'); % Average line
grid on;

xlabel('Budget Spent on Weight (£)');
ylabel('Lap Time (s)');
title('Lap Time vs Budget Spent');
legend('show');
hold off;

% Plot weight and ride height lap times on a separate graph
figure;
plot(budget_weight, lap_times_weight, 'b-', 'LineWidth', 2, 'DisplayName',
'Weight Reduction');
hold on;
plot(budget_height, reversed_lap_times_height, 'g-', 'LineWidth', 2,
'DisplayName', 'Ride Height Adjustment (Reversed)');
```

```
grid on;

% Calculate and plot the average line between the two adjustments
% Interpolation to get average lap time for the combined budget
combined_budget = linspace(0, budget_limit, 100);
combined_lap_times = zeros(size(combined_budget));

for i = 1:length(combined_budget)
    budget = combined_budget(i);
    % Find corresponding lap times from weight and ride height arrays
    weight_idx = find(budget_weight <= budget, 1, 'last');
    height_idx = find(budget_height <= budget, 1, 'last');

    % If valid indexes are found, average the lap times
    if ~isempty(weight_idx) && ~isempty(height_idx)
        combined_lap_times(i) = (lap_times_weight(weight_idx) +
reversed_lap_times_height(height_idx)) / 2;
    end
end

% Plot the combined average lap time line
plot(combined_budget, combined_lap_times, 'm--', 'LineWidth', 2,
'DisplayName', 'Combined Average Lap Time'); % Average line
xlabel('Budget Spent on Weight (£)');
ylabel('Lap Time (s)');
title('Lap Time vs Budget Spent for Individual and Combined Adjustments');
legend('show');
hold off;
```


Lap Time vs Budget Spent
Lap Time
Average Lap Time
Lap Time (s)
94
92
90
88
86
84
82
80
0
0.5
1
1.5
2
2.5
Budget Spent on Weight (£)
×10^4


Lap Time vs Budget Spent for Individual and Combined Adjustments
Weight Reduction
Ride Height Adjustment (Reversed)
Combined Average Lap Time
Lap Time (s)
94
92
90
88
86
84
82
80
0
0.5
1
1.5
2
2.5
Budget Spent on Weight (£)
×10^4

Helper Function for Lap Time Calculation

```
function lap_time = calculate_lap_time(mass, drag_coeff, tire_mu, power, ...
    num_turns, corner_lengths, avg_straight_length, g, air_density,
frontal_area, mass_original)
    lap_time = 0;

    % Cornering Time
    for j = 1:num_turns
        radius = 30+j*5; % Example radius
        effective_mu = tire_mu * (mass_original / mass); % Adjust tire mu
based on mass change
        max_cornering_speed = sqrt(effective_mu * g * radius); % Adjust
based on effective tire mu
        corner_time = corner_lengths(j) / max_cornering_speed; % Time to
complete the corner
        lap_time = lap_time + corner_time;
    end

    % Straight-Line Time
    power_to_weight = power * 1000 / mass; % Power-to-weight ratio (in Watts
per kg)
    straight_accel = power_to_weight; % Acceleration (simplified model)
    top_speed = sqrt((2 * power * 1000) / (drag_coeff * air_density *
frontal_area)); % Top speed

    for j = 1:(num_turns - 1)
        d_accel = min((top_speed^2) / (2 * straight_accel),
avg_straight_length);
        t_accel = sqrt(2 * d_accel / straight_accel); % Time for
acceleration phase

        if d_accel < avg_straight_length
            d_cruise = avg_straight_length - d_accel;
            t_cruise = d_cruise / top_speed;
        else
            t_cruise = 0;
        end
        lap_time = lap_time + t_accel + t_cruise;
    end
end
```

Published with MATLAB® R2024b