Specify Purpose

This model aims to predict the optimal spacing of yellow transverse bar markings[1] painted on a main carriageway that alert a driver to a change of speed limit via the visual, auditory, and sensory cues they provide.

Create the Model

Describe features investigated and outline mathematics used.

We model the speed of the car between the first and the last transverse bar. These will be the main inputs of the model. We aim to utilise the deceleration effect of said markings to best effect. The outputs will be the width between each successive bar and the distance of each bar from some reference point. We consider the condition of the road, the behaviour of the driver and the size and shape of the vehicle.

State Assumptions

1. We ignoring air resistance and friction and model the vehicle as a particle.
2. The road is perfectly straight with no gradient, clear of other vehicles and has a uniform surface.
3. The driver of the vehicle will take time to react before braking.
4. The transverse bars are lines of zero width.
5. The vehicle accelerates only within the confines of the transverse bars.
6. The vehicle travels at the speed limits on approach to and exit from the transverse bars.
7. The vehicle will brake to ensure the bars are passed at a constant rate.

[1] UK Government traffic signs manual, chapter 5; ‘Road Markings’ pages 71-73, (2019) https://www.gov.uk/government/publications/traffic-signs-manual

Variables & Parameters

Symbol	Description	Unit	Notes
u	Velocity at $x = 0$ and $x = x_0$	$\mathrm{m\ s^{-1}}$	Speed limit of the main road. Input
v	Velocity at $x = D$	$\mathrm{m\ s^{-1}}$	Speed limit of the slip road. Input
a_0	Constant acceleration	$\mathrm{m\ s^{-2}}$	Derived quantity
n	Total number of transverse bar markings	1	$n \in \mathbb{Z}^+$
D	Distance between bar 1 and bar n	m	Input
D_n	Position of transverse bar relative to $D_1 = 0$	m	$n \in [1, n]$
d_k	Distance between each transverse bar	m	$k \in [1, n-1]$
x_0	Distance travelled during the ‘Thinking’ phase	m	
T	Total time between $D_1\ and\ D_n$	s	
δt	Constant time interval during ‘Thinking’ phase	s	Derived quantity
δt_2	Constant time interval during ‘Acceleration’ phase	s	Derived quantity
t_R	Driver reaction time	s	Input

Table 1

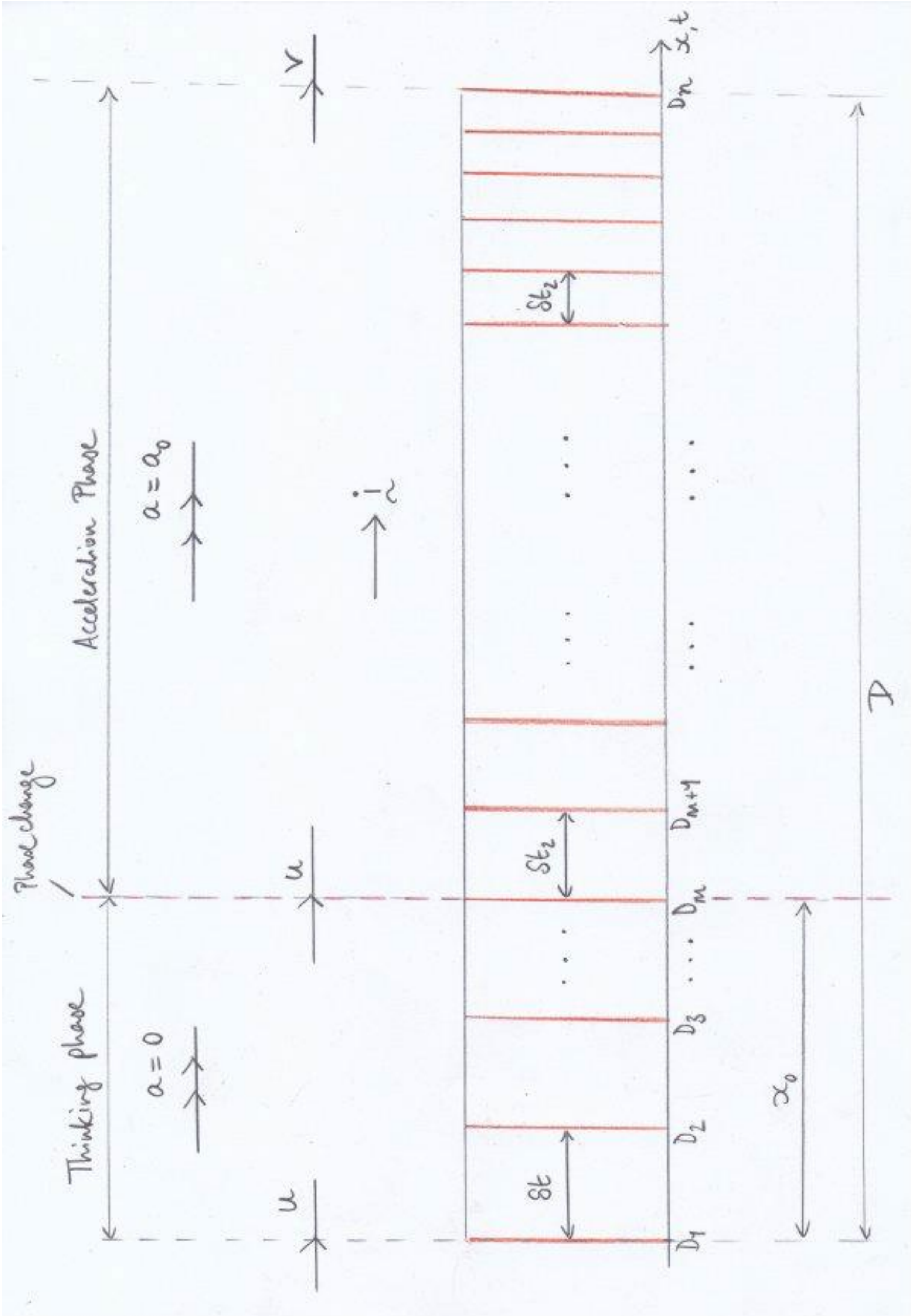

Thinking phase
Acceleration Phase
Phase change
$a = 0$
$a = a_0$
u
u
v
x, t
δt
δt_2
δt_2
D_1
D_2
D_3
D_m
D_{m+1}
D_n
x_0
D

Figure 1

Formulate mathematical relationships.

In Figure 1 by assumptions (1), (2) & (6), the particle is moving in a straight line on a smooth surface in the positive x-direction.

At time $t = 0$, its position is $x\mathbf{i} = 0$ with velocity $v_0\mathbf{i} = u$.

By assumption (7) at some time t, its velocity and position are given by the equations,

$$v = u + a_0 t \tag{1}$$

$$x = x_0 + ut + \frac{1}{2}a_0 t^2 \tag{2}$$

$$v^2 = u^2 + 2a_0 D \tag{3}$$

MST210 HB. pg. 39

We derive the total time interval from eq.(1),

$$T = \frac{v - u}{a_0} \tag{4}$$

We derive the constant acceleration from eq.(3),

$$a_0 = \frac{v^2 - u^2}{2D} \tag{5}$$

Combining the above equations gives,

$$T = \frac{(v - u)2D}{(v + u)(v - u)} = \frac{2D}{v + u}$$

By assumptions (4), (5) and (7), the driver passes the lines at a constant rate δt.

$$T = (n - 1)\delta t$$

$$\delta t = \frac{2D}{(v + u)(n - 1)} \tag{6}$$

Do the Mathematics

Deriving a first model

Thinking time phase, $x = 0$ to $x = x_0$

By assumption (3) in eq.(2), whilst thinking about braking, the driver will cover a distance,

$$x_0 = ut_R$$

t_R seconds contains $t_R/\delta t$ intervals of δt rounded to the nearest integer m.
We have the recurrence relation,

$$D_1 = 0, \qquad D_n = D_{n-1} + \frac{x_0}{m} \qquad (n = 2, 3, \dots, m+1)$$

With closed form,

$$D_n = (n-1)\frac{x_0}{m} \quad (n = 1, 2, \dots, m+1) \tag{7}$$

Acceleration phase, $x = x_0$ to $x = D$

By assumption (3), the acceleration from $x = x_0$ to $x = D$ at $t = T$ is given,

$$a = a_0$$

During this constant acceleration phase, we require a new constant time interval δt_2.
From eq.(6),

$$\delta t_2 = \frac{2(D - x_0)}{(v + u)((n - m) - 1)}$$

By assumption (4), we assume the lines are of the zero width.
The position of the n^{th} line given by eq.(2),

$$D_n = x_0 + u\delta t_2 k + \frac{1}{2}a(\delta t_2 k)^2 \tag{8}$$

Where $n \in [5, n]$ and $k \in [1, (n - m - 1)]$

Which is given explicitly as,

$$D_n = ut_R + \frac{2(D - x_0)u}{(v + u)(n - m - 1)}k + \frac{(D - x_0)(v^2 - u^2)}{(v + u)^2(n - m - 1)^2}k^2$$

The distance d_k between each line D_n is given by the difference between successive terms in the sequence. That is,

$$d_k = D_{n+1} - D_n \qquad n \in [1, n - 1] \text{ and } k \in [1, n - 1] \tag{9}$$

Equation (8) returns a sequence of lengths in relation to the reference point $x = 0$ that represent the position of the transverse bars where $D_1 = 0$ through to $D_n = D$

Equation (9) also returns a sequence of lengths representing the distance between each successive bar.

Graphs of typical relationships

Distance/Time Graphs

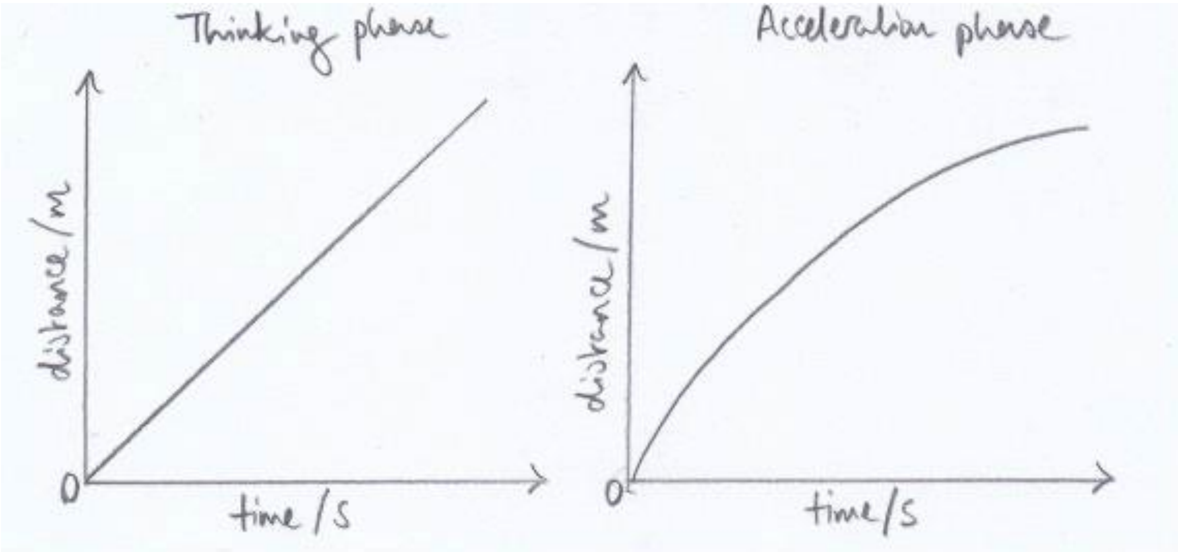


Figure 2

From eq.(8) we would expect the distance/time graph to be a straight line for the 'thinking distance' phase and a slight 'n' shaped parabola for the 'constant acceleration' phase.

Velocity/Time Graphs

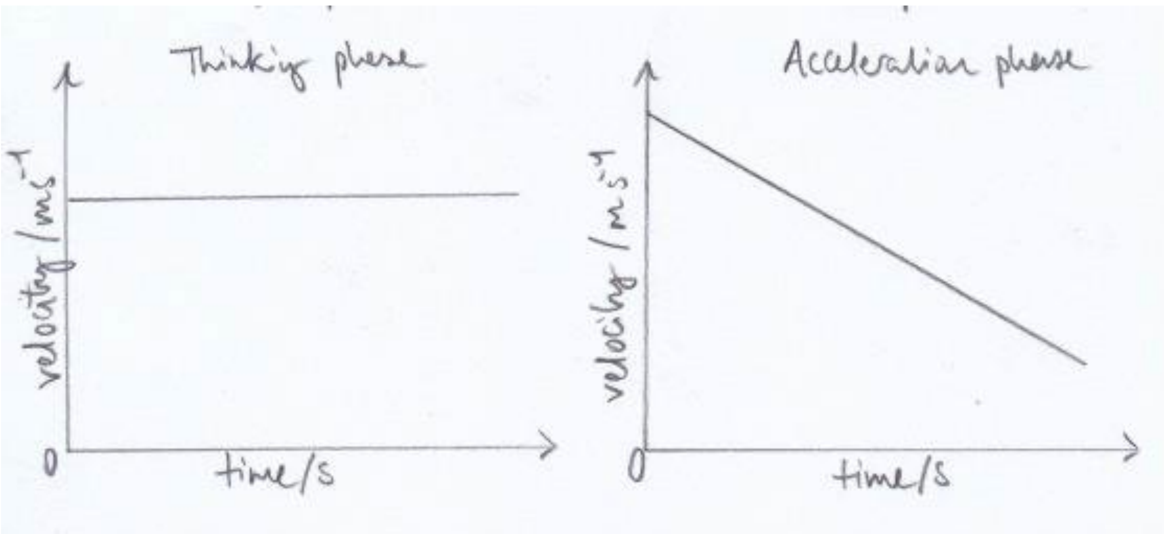


Figure 3

The derivative of eq.(8) is a linear function. Subject to a constant negative acceleration, velocity decreases over time as a straight line

Acceleration/Time Graph

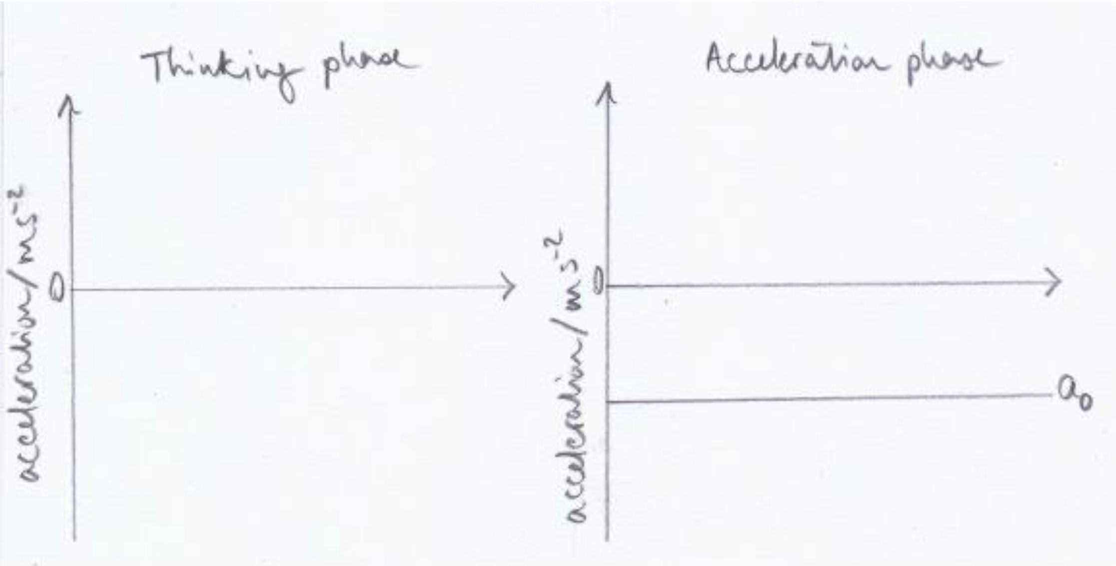


Figure 4

The derivative of the velocity function returns constant acceleration. The acceleration/time graph is a constant negative acceleration as it represents the vehicle braking.

Dimensional Analysis

We analyse the dimensions of the explicit form of eq.(8).

$$D_n = ut_R + \frac{2(D - x_0)u}{(v + u)(n - m - 1)}k + \frac{(D - x_0)(v^2 - u^2)}{(v + u)^2(n - m - 1)^2}k^2$$

The LHS of the above equation represents a length with dimension $[L]$
Term by term, the RHS of eq.(8) gives,

$$[ut_R] = [u][t_R] = (L\,T^{-1}) \times T = L$$

$$\left[\frac{2(D - x_0)u}{(v + u)(n - m - 1)}k\right] = \left[\frac{[2]([D] - [x_0])[u]}{([v] + [u])([n] - [m] - [1])}[k]\right]$$

$$= \left[\frac{1 \times L \times (L\,T^{-1})}{\big((L\,T^{-1}) + (L\,T^{-1})\big) \times 1}[1]\right] = L \times (L\,T^{-1}) \times (L^{-1}\,T) = L$$

$$\left[\frac{(D-x_0)(v^2-u^2)}{(v+u)^2(n-m-1)^2}k^2\right]=\left[\frac{([D]-[x_0])([v]^2-[u]^2)}{([v]+[u])^2([1]-[1]-[1])^2}[1]^2\right]$$

$$=\left(\frac{(L-L)((L\,T^{-1})^2-(L\,T^{-1})^2)}{(L\,T^{-1}+L\,T^{-1})^2\times 1^2}\right)(1)^2$$

$$=L\times(L^2\,T^{-2})\times(L^{-2}\,T^2)\times 1=L$$

Each term in the LHS of eq.(8) has dimension L. The equation is dimensionally consistent.

Interpret the results.

Collect relevant data for parameter values.

According to the UK government[2], the 70-mph national speed limit applies to all dual carriageways and motorways. A 30-mph speed limit applies to built-up areas and roads with streetlights. Our model considers a 'main carriageway' therefore, we assume the bars are designed to accommodate the national speed limit as opposed to any local variable speed restrictions.

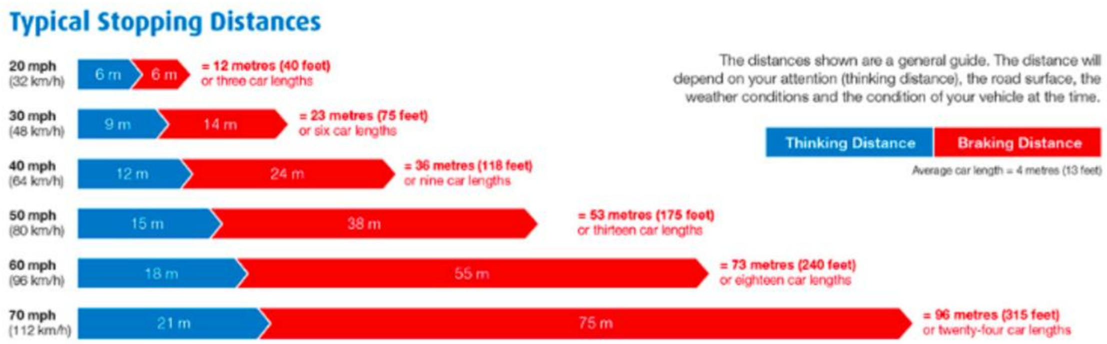
Figure 5

[2] https://www.gov.uk/speed-limits,

Figure 5 shows junction 1 of the M1 viewed from Google Earth, the transverse bars cover a distance of approximately 400 m where on counting the bars, there are 90.
We assume the national speed limit applies to the dual carriageway leading to junction 1 of M1. We assume the roundabout leading to the east bound A406, and Brent Cross shopping centre has a 30-mph speed limit.

Thinking time

Section 126, pg 83 of the Highway code[3] recommends the following thinking distances.



Where from eq.(2) and the above figure,

$$70 \text{ mph}\,; \;\; 21 \text{ m} = \; 31.293 \text{ m s}^{-1} \times t \rightarrow t = \frac{21}{31.293} = 0.671 \text{ s}$$

The initial parameters to test our model are,

Parameter	Values	SI conversion
u	70 mph	31.293 m s^{-1}
v	30 mph	22.352 m s^{-1}
D	400 m	
n	90 bars	
t_R	0.67 s	

Table 2

Miles per hour to metres per second conversion[4]

[3] https://www.highwaycodeuk.co.uk/download-pdf.html

[4] Appendix 1.1

Describe the mathematical solution.

From the above parameters, the output of our model will be an arithmetic sequence of n distances from the reference point $D_1 = x = 0$.
The derived equations have been encoded into Python and multiple functions[5] have been defined. Full details of the calculations can be found in this GitHub Repository

Find predictions to compare with reality.

[5] Appendix 1.2

Evaluate the Model

Evaluate the model **15 marks**

Collect data to compare with the model. Collect additional data to test your model. Do not use the data used to define parameter values. Usually, the additional data will be from the internet (or the library) and should be referenced. If your data are from a simple experiment, then state the results here and describe the experiment in an appendix.

Test your first model. Compare model predictions with your additional data. Some models may be impossible to test in this way, in which case you should explain why it is not possible to test your model. Marks are available for describing a test without actually being able to perform it.

Criticise your first model. Criticise your model based on the tests that you performed.

Review your assumptions. Consider each assumption in turn, and explain what would be the effect of changing it. Focus on those assumptions that would improve the fit to the evaluation data.

Revise the model.

Revise the model **10 marks**

Decide whether to revise your first model. Decide whether a revision of your first model is justified. Explain why you made your decision, referring to the evaluation of the first model and your review of the assumptions. If your first model fits your data well, then consider if a simpler model might be better.

Describe your intended revision. Include a clear statement of any assumptions that are being revised and the new assumption(s) that will replace them. Note that a change of a parameter value does not constitute a revision of the model. Try to explain how the revision you suggest might affect any differences between the predictions of the first model and the data used for evaluation.

Conclusions

Conclusions **5 marks**

Summarise your modelling. Include the performance of your first model, any attempts to improve on it, and any comments on the modelling process. This short summary should not introduce any new considerations.

Appendix

1.1

Miles per hour to metres per second conversion

$$1 \text{ mile} = 1.609344 \times 10^3 \text{ m}$$
$$1 \text{ hour} = 60 \times 60 \text{ s}$$

$$\therefore x \text{ mph} = x \text{ miles h}^{-1} \times \left(\frac{1.609344 \times 10^3 \text{ m}}{1 \text{ mile}}\right) \times \left(\frac{60 \times 60 \text{ s}}{1 \text{ hr}}\right)^{-1}$$
$$= y \text{ m s}^{-1}$$

Or

$$\text{m s}^{-1} = \text{mph} \times 0.44704$$

1.2

Raw python code

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def metres_per_second(u, v):
    initial_velocity = round(u * 0.44704, 3)
    final_velocity = round(v * 0.44704, 3)

    return print(f"{u} mph is {initial_velocity} metres per second\n"
                 f"{v} mph is {final_velocity} metres per second")


def calculate_distances(u: float, v: float, D: float,
                        n: int, thinking_time=0.0,
use_thinking_distance=False) -> list:
    """
    This function calculates the position of the transverse bars
    Takes parameters;
    u = initial velocity in metres per second
    v = final velocity in metres per second
    D = total distance in metres
    n = number of bars
    thinking_time in seconds
    use_thinking_distance, set to false for no thinking time
    """
    distances = [0] * n

    x_0 = u * thinking_time
    a = (v**2 - u**2) / (2*D)
```

```
    t = (v - u) / a
    delta_t = t / (n-1)
    x_0_bars = thinking_time / delta_t

    # Calculate the distances for the first n_0 time intervals, where the
speed is constant
    n_0 = int(round(x_0_bars)) # Number of time intervals during which the
speed is constant

    for i in range(1, n):
        if use_thinking_distance:
            const_vel = [x_0 / n_0] * n_0
            for i in range(1, n_0):
                const_vel[i] += const_vel[i-1]
                distances[1:n_0+1] = const_vel
                # Calculate distances for remaining time intervals, where
constant acceleration applies
            d_remaining = D - x_0
            a = (v**2 - u**2) / (2*d_remaining)
            t = (v - u) / a
            delta_t = t / (n-(n_0+1))

            for i in range(n_0, n):
                t_i = delta_t * (i - n_0) # Time since the end of constant
velocity phase
                distances[i] = x_0 + (u * t_i) + (0.5 * a * (t_i ** 2))
                # Round distances to three decimal places
        else:
            distances = [a*(delta_t*k)**2/2 + u*(delta_t*k) for k in
range(n)]

            #dk_values.insert(0, x_0) # insert x_0 at the beginning of the
list
            #distances.append(D) # append d at the end of the list
    distances = [round(x, 3) for x in distances]
    return distances


def distance_dk(list: list) -> list:
    """
    Take a list of the bar markings in relation to x=0
    returns the distance between each bar
    :param list:
    :return: list
    """
    # Calculate the list of distances between consecutive markers
    distances = [list[0]] + [list[k] - list[k-1] for k in range(1,
len(list))]
    distances = [round(x, 3) for x in distances]

    # We get rid of the zero element
    distances = distances[1:len(distances)]

    return distances

def plot_subgraphs_dk(no_thinking_time_list: list, thinking_time_list:
list, real_world_list: list) -> plt:
    """
    Takes either;
        new_distances for n=90
```

```
        new_distances2 for n=45
    for real_world_list parameter
    :param no_thinking_time_list:
    :param thinking_time_list:
    :param real_world_list:
    :return: figure object containing two subplots
    """
        # Create figure with two subplots
    fig, axs = plt.subplots(1, 2, figsize=(10, 4))

    # No Thinking time
    n = len(no_thinking_time_list) + 1
    axs[0].plot(np.arange(1, n), no_thinking_time_list)
    axs[0].plot(np.arange(1, n), real_world_list)
    axs[0].set_title('No Thinking Time', fontsize='small')
    axs[0].set_xlabel('$d_k$')
    axs[0].set_ylabel('Distance between consecutive markers / m')
    x_ticks = np.arange(0, n, 10)
    axs[0].set_xticks(x_ticks)
    axs[0].legend(['Model', 'Real World'])

    # Thinking Time
    n = len(real_world_list) + 1
    axs[1].plot(np.arange(1, n), thinking_time_list)
    axs[1].plot(np.arange(1, n), real_world_list)
    axs[1].set_title('With Thinking Time', fontsize='small')
    axs[1].set_xlabel('$d_k$')
    axs[1].set_ylabel('Distance between consecutive markers / m')
    x_ticks = np.arange(0, n, 10)
    axs[1].set_xticks(x_ticks)
    axs[1].legend(['Model', 'Real World'])

    fig.subplots_adjust(wspace=0.3)
    fig.suptitle("Distance $d_k$ between each successive $D_n$",
fontsize='large')

    #filename = f"sidebyside_dk_n={n}.png"
    #plt.savefig(filename)

    return plt.show()


def plot_graph_dk(model_list: list, real_world_list: list) -> plt:
    """
    Takes two lists for model and real-world distances and plots them on a
single graph.
    :param model_list:
    :param real_world_list:
    :return: figure object containing a single plot
    """
    # Create figure with a single plot
    fig, ax = plt.subplots(figsize=(8, 6))

    n = len(model_list) + 1
    ax.plot(np.arange(1, n), model_list)
    ax.plot(np.arange(1, n), real_world_list)
    ax.set_title(f"Distance $d_k$ between each successive $D_n$\n$k={n}$",
fontsize="large")
    ax.set_xlabel("$d_k$")
    ax.set_ylabel("Distance between consecutive markers / m")
```

```
    x_ticks = np.arange(0, n, 10)
    ax.set_xticks(x_ticks)
    ax.legend(["Model", "Real World"])

    #filename = f"single_dk_n={n}.png"
    #plt.savefig(filename)

    return plt.show()

def plot_single_dk(model_list: list) -> plt:
    """
    :param model_list:
    :return: figure object containing a single plot
    """
    # Create figure with a single plot
    n = len(model_list)
    plt.plot(np.arange(1, n+1), model_list)
    plt.title(f"Distance $d_k$ between each successive $D_n$\n$k={n}$",
fontsize="large")
    plt.xlabel("$d_k$")
    plt.ylabel("Distance between consecutive markers / m")
    x_ticks = np.arange(0, n+1, 5)
    plt.xticks(x_ticks)
    #filename = f"single_dk_n={n}.png"
    #plt.savefig(filename)

    return plt.show()


def get_delta_t(u: float, v: float, D: float,
               n: int, thinking_time=0.0, use_thinking_distance=False):
    """
    Takes similar parameters to calculate_distances function
    returns delta_t's
    delta_t2 = 0.0 if use_thinking_distance = default = False
    """
    x_0 = u * thinking_time
    a = (v**2 - u**2) / (2*D)
    t = (v - u) / a
    delta_t1 = t / (n-1)
    x_0_bars = thinking_time / delta_t1
    n_0 = int(round(x_0_bars))

    if use_thinking_distance:
        d_remaining = D - x_0
        a2 = (v**2 - u**2) / (2*d_remaining)
        t2 = (v - u) / a2
        delta_t2 = t2 / (n-(n_0+1))
    else:
        delta_t2 = 0.0

    delta_t1 = round(delta_t1, 6)
    delta_t2 = round(delta_t2, 6)

    print(f"Delta_t1: {delta_t1}\nDelta_t2: {delta_t2}")
    return delta_t1, delta_t2


def distance_time_graph(u: float, v: float, D: float, n: int,
                thinking_time=0.0, use_thinking_distance=False):
```

```
    """
        Takes parameters;
    u = initial velocity in metres per second
    v = final velocity in metres per second
    D = total distance in metres
    n = number of bars
    thinking_time in seconds
    use_thinking_distance, set to false for no thinking time
    delta_t2 = 0.0 if use_thinking_distance = default = False

    returns list of 2D vectors
    """
    distances_list = calculate_distances(u, v, D, n, thinking_time,
use_thinking_distance)
    graph_points = []

    n = len(distances_list)
    x_0 = u * thinking_time
    a = (v**2 - u**2) / (2*D)
    t = (v - u) / a
    delta_t1 = t / (n-1)
    x_0_bars = thinking_time / delta_t1
    n_0 = int(round(x_0_bars))

    if use_thinking_distance:
        d_remaining = D - x_0
        a2 = (v**2 - u**2) / (2*d_remaining)
        t2 = (v - u) / a2
        delta_t2 = t2 / (n-(n_0+1))
    else:
        delta_t2 = 0.0

    for i in range(n):
        if i <= n_0:
            t_i = round(i * delta_t1, 3)
        else:
            if use_thinking_distance:
                t_i = round((n_0 * delta_t1) + ((i - n_0) * delta_t2), 3)
            else:
                t_i = round(i * delta_t1, 3)
        graph_points.append([t_i, distances_list[i]])
        if i == n_0 and use_thinking_distance:
            delta_t1 = delta_t2

    # Extract the time and distance values from the graph points
    times = [point[0] for point in graph_points]
    distances = [point[1] for point in graph_points]

    if use_thinking_distance:
        string = "With Thinking Time"
    else:
        string = "No Thinking Time"
    # Plot the graph using the time and distance values
    plt.plot(times, distances)
    plt.title(f"Distance/Time Graph\n{string}, $n={n}$")
    plt.xlabel('Time (s)')
    plt.ylabel('Distance (m)')

    return plt.show(), print(len(graph_points), graph_points)
```

```
def velocity_time_graph(u: float, v: float, D: float, n: int,
thinking_time=0.0, use_thinking_distance=False):
    """
    This function creates a velocity time graph
    Takes parameters;
    u = initial velocity in metres per second
    v = final velocity in metres per second
    D = total distance in metres
    n = number of bars
    thinking_time in seconds
    use_thinking_distance, set to false for no thinking time

    Returns velocity/time graph
    """

    # Equations of motion
    a = (v**2 - u**2) / (2*D)
    t = (v - u) / a
    delta_t1 = t / (n-1)
    x_0 = u * thinking_time
    x_0_bars = thinking_time / delta_t1
    n_0 = int(round(x_0_bars))

    for i in range(1,n):
        if use_thinking_distance:
            d_remaining = D - x_0
            a2 = (v ** 2 - u ** 2) / (2 * d_remaining)
            t2 = (v - u) / a2
            delta_t2 = t2 / (n - (n_0 + 1))
            v_list_const = [u for n in range(0,n_0)]
            v_list_acc = [u + a * delta_t2 * n for n in range(n_0, n)]
            vn_values = v_list_const + v_list_acc
        else:
            vn_values = [u + a * delta_t1 * n for n in range(0, n)]

    vn_values = [round(x, 3) for x in vn_values]

    num_points = len(vn_values)
    if use_thinking_distance:
        time_list = [round(delta_t1 * i, 3) if i <= n_0 else round(delta_t2
* i, 3)for i in range(n)]
    else:
        time_list = [delta_t1 * n for n in range(num_points)]


    # plot d over time
    if use_thinking_distance:
        string = f"With Thinking Time, $n={n}$"
    else:
        string = f"No Thinking Time, $n={n}$"
    plt.plot(time_list, vn_values)
    plt.xlabel('Time / s')
    plt.ylabel('Velocity / $\mathrm{m\ s{^{-1}}}$')
    plt.title(f'Velocity vs. Time\n{string}')
    if use_thinking_distance:
        title = f"vel_time_thinking_n={n}"
    else:
        title = f"vel_time_no_thinking_n={n}"
```

```
    #plt.savefig(f'{title}.png')

    return plt.show()


def dataframe(real_world_list: list, thinking_list: list, no_thinking_list:
list):
    """

    """

    d_k = [f"d{i}" for i in range(1, len(real_world_list)+1)]
    dk_df = pd.DataFrame({'d_k': d_k, 'real_world': real_world_list,
                          'thinking': thinking_list, 'no_thinking':
no_thinking_list})

    return dk_df


def stats(Dataframe):
    """
    Take a pandas DataFrame as input

    returns: summary statistics and boxplot
    """
    stats = Dataframe.describe()

    Dataframe.boxplot(column=['real_world', 'thinking', 'no_thinking'])
    plt.title("Box plot of real world vs. model data")
    plt.ylabel("Distance / metres")

    if len(Dataframe['real_world'].tolist()) > 45:
        string = "90 Transverse bars"
    else:
        string = "45 Transverse bars"

    return print(string), plt.show(), print(f"\nSummary
Statistics:\n\n{stats}")
```

Bibliography