

Truck Platooning

Stephanie Chinenye Okosa Maxim Emile Speczyk Muhammad Umer Bin Yaqoob¹

Contents

1 Motivation	2
2 Foundation	2
2.1 Requirement Diagram	3
2.2 State Machine Diagrams	3
2.2.1 Braking Scenario	3
2.2.2 Steering Scenario	4
2.2.3 Lane Change Scenario	4
3 UPPAAL IMPLEMENTATION	4
3.1 UPPAAL Braking Modelling	5
3.2 UPPAAL Steering Modelling	5
3.3 UPPAAL Lane Change Modelling	5
3.4 UPPAAL Truck Coupling and Decoupling Modelling	6
4 Machine Learning Algorithm	6
4.1 Decision Tree Classifier	7
4.2 Isolation Forest Model	7

¹ stephanie-chinenye.okosa@stud.hshl.de, muhammad-umer-bin.yaqoob@stud.hshl.de, maxim-emile.speczyk@stud.hshl.de

5	Truck Platoon Scenarios Simulation	8
5.1	Braking Simulation	8
5.2	Steering Simulation	9
5.3	Lane Change Simulation	9
6	Declaration of Originality	10

Abstract: This project presents an autonomous truck platooning system's design, implementation, and testing. This technological solution promises to revolutionise the freight industry by increasing efficiency and safety while reducing costs. The system, modelled using UML, incorporates a machine learning algorithm to select the platoon leader based on multiple parameters. It considers various scenarios like steering, lane change, braking, joining, and leaving the platoon, including potential communication failures implemented in UPPAAL. The project will culminate in a comprehensive simulation environment using Python.

1 Motivation

Autonomous vehicles have revolutionised our society, bringing with them a myriad of opportunities and challenges. Our team has embarked on a project to develop an autonomous truck platooning system that can simulate the steering, braking, lane change, coupling and decoupling of the trucks during motion. Truck platooning essentially involves several trucks equipped with advanced driver assistance systems. These trucks follow each other closely on a motorway, literally forming a convoy or 'platoon'. The importance of this arrangement lies in the many benefits it offers which include; improved road safety, reduced fuel consumption, reduced aerodynamic drag, lower CO_2 emissions and more efficient traffic flow.

2 Foundation

In the early stages of our project, each member of our team brainstormed and generated detailed scenarios for our autonomous truck platooning system. This step helped us to identify key functional requirements and gain a deep understanding of the problem from the user's perspective. We then used UML (Unified Modelling Language), to visually represent these scenarios and the system as a whole, detailing the relationships and data flow between entities. The UML designs served as blueprints for the architecture of our system, providing a solid foundation for model specification and final implementation, while facilitating efficient communication and shared understanding among team members and stakeholders.

2.1 Requirement Diagram

Initial requirements of the truck platooning system were discussed, the following requirements were iterated, and identified as critical for the system to function properly as shown in [1] below.

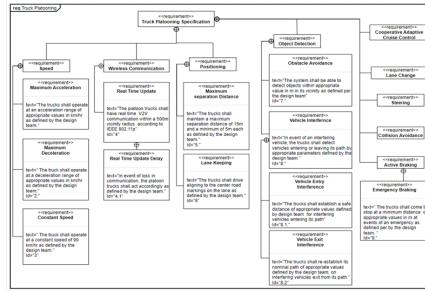


Fig. 1: Requirement Diagram

<https://github.com/MaximSpeczyk/Autonomous-Lab-A-Group-4/blob/main/requirements/truckPlatooningRequirements.png>

2.2 State Machine Diagrams

In this section we show the different scenarios states to enable us model the the timing behaviour in UPPAAL.

2.2.1 Braking Scenario

The autonomous truck platooning project focuses on synchronised braking mechanisms across the fleet. When the master truck initiates braking, every other truck must follow suit to maintain safe following distances and avoid collisions. This is achieved through efficient communication paths between the lead and following trucks as shown in [2]below.

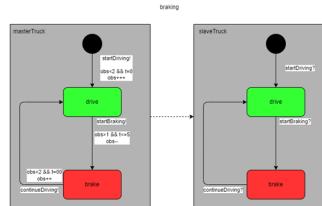


Fig. 2: Braking Scenario

<https://github.com/MaximSpeczyk/Autonomous-Lab-A-Group-4/blob/main/state%20Machine/braking.png>

2.2.2 Steering Scenario

This scenario in [3] ensures that the subordinate, or 'slave', trucks follows the lead or 'master' truck within the platoon at real time by, implementing a communication protocol wherein the leading, or 'master', truck transmits directional commands to the follower trucks. This facilitates seamless and synchronized transitions between directional states, e.g., initiating rightward or leftward shifts. The behaviour of this brake control system is illustrated by a state machine diagram in our system design.

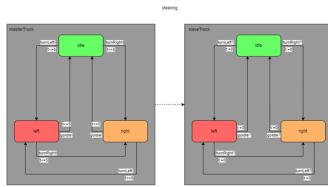


Fig. 3: Steering Scenario

<https://github.com/MaximSpeczyk/Autonomous-Lab-A-Group-4/blob/main/state%20Machine/steering.png>

2.2.3 Lane Change Scenario

Lane change coordination is key within the truck platoon fleet. The aim is to ensure that when the master truck makes a lane change, the slave trucks systematically reproduce the action to maintain the integrity of the platoon. These lane changes are managed by a highly responsive communication system between the trucks as shown in [4] below.

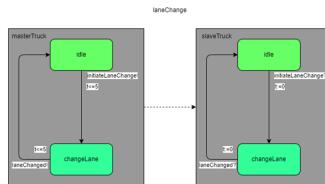


Fig. 4: Lane Change Scenario

<https://github.com/MaximSpeczyk/Autonomous-Lab-A-Group-4/blob/main/state%20Machine/laneChange.png>

3 UPPAAL IMPLEMENTATION

Here data, signal, events that are required for interactive communication between the trucks are identified, appropriate protocol for each scenario using timed automata specified, and modelled using UPPAAL. All scenarios described in this section model a platoon system where one truck acts as the 'truckMaster' and the two slave trucks, 'truckSlave' follow.

3.1 UPPAAL Braking Modelling

Here, the master truck leads the platoon, broadcasting driving instructions via three primary channels; 'continueDriving', 'Brake', and 'continueBraking'. Depending on these signals, the slave trucks respond by maintaining speed, applying brakes, or continuing to brake. This synchronization enables the platoon to operate harmoniously and safely.

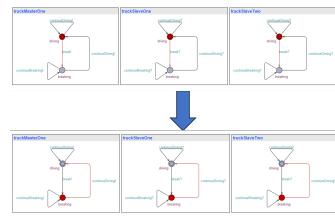


Fig. 5: UPPAAL Braking
<https://github.com/MaximSpeczyk/Autonomouse-Lab-A-Group-4/blob/main/scenarios/braking.xml>

3.2 UPPAAL Steering Modelling

The steering behaviour of both trucks is defined by three states: idle, turnLeft and turnRight. The master truck's state transitions are determined by a global clock and its own synchronisation events, while the slave truck mimics the master's steering actions. The model thus demonstrates a coordinated steering system in a truck platoon, ensuring that the slave truck accurately replicates the directions of the master.

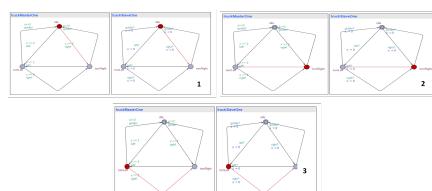


Fig. 6: UPPAAL Steering
<https://github.com/MaximSpeczyk/Autonomouse-Lab-A-Group-4/blob/main/scenarios/steering.xml>

3.3 UPPAAL Lane Change Modelling

Lane change is initiated by the lead truck after a set time, prompting the following trucks to begin the same procedure. A 'Watchdog' entity monitors the operation, ready to raise an alarm if the lane change takes longer than expected. Upon completion, the lead truck signals the end of the lane change, upon which all trucks resume cruising. This system enables safe and synchronized lane changes within a truck platoon.

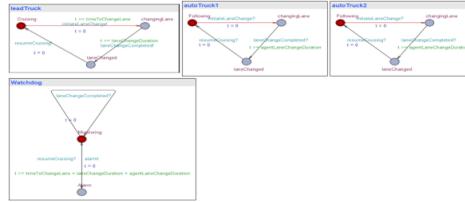


Fig. 7: UPPAAL Lane Change

<https://github.com/MaximSpeczyk/Autonomouse-Lab-A-Group-4/blob/main/scenarios/laneChanging.xml>

3.4 UPPAAL Truck Coupling and Decoupling Modelling

In this scenario, each truck initially operates independently until it identifies a suitable environment for platooning, such as a motorway. When it detects a potential platoon, the truck sends a request to join, which is evaluated by the existing platoon, taking into account factors such as the size of the platoon, the compatibility of the new truck, and the potential benefits and risks. If the request is approved, the truck joins the platoon, adjusting its speed and positioning to form a coordinated and efficient transport unit. Critical to this operation is the constant communication between the trucks, which is monitored by a watchdog unit that ensures consistent data exchange and monitors for potential anomalies. If the watchdog detects a communication disruption within a truck that exceeds a certain threshold, the affected truck is removed from the platoon until the problem is resolved. In the event of a major malfunction affecting the entire platoon, the platoon disbands and each truck returns to independent operation, demonstrating the resilience and adaptability of the system.

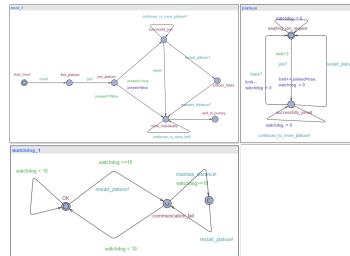


Fig. 8: UPPAAL Coupling and Decoupling

<https://github.com/MaximSpeczyk/Autonomouse-Lab-A-Group-4/blob/main/scenarios/couplingAndDecoupling.xml>

4 Machine Learning Algorithm

To determine the lead vehicle of the platoon for simulating the various scenarios, machine learning algorithm is applied on data information of the trucks with predefined features to

train our model to autonomously identify the 'truckMaster' for each platoon. Two different methods are applied in order to determine the algorithm with a better prediction accuracy.

4.1 Decision Tree Classifier

Here, the decision tree classifier is used for classification of the 'truckMaster' from the 'truckSlave' in the platoon using a set of predefined features as shown in [9]. However the prediction score is 50% therefore, we try the isolation forest model to improve the prediction accuracy of the model.

```

distance_route    max_route_match   fuel_consumption   body_characteristics   equipment_sensors   leader_vehicle
0          808           100            245             3769              91                0
1          899            42             308             2732              82                1
2          889            43             237             3876              72                0
3          823            35             303             2934              73                0
4          997            50             253             3649              57                0

# Define features and target
x = data.drop(['leader_vehicle'], axis=1)
y = data['leader_vehicle']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)

# Create a DecisionTreeClassifier and fit it to the training data
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3)
clf = clf.fit(x_train, y_train)

# Predict the test set results
y_pred = clf.predict(x_test)

# Print the accuracy of the model
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

# Plot the decision tree
plt.figure(figsize=(15,15))
clf = DecisionTreeClassifier(max_depth=3).fit(x, y)
dot_data = export_graphviz(clf, filled=True, fontsize=10)
plt.show()

```

Fig. 9: Decision Tree Classifier

<https://github.com/MaximSpeczyk/Autonomouse-Lab-A-Group-4/blob/main/Leader%20vehicle%20selection/decisionTreeClassifier/DecisionTreeClassifier.py>

4.2 Isolation Forest Model

The isolation forest model is based on decision trees and is used in identifying anomalies in a data set by employing binary trees. The idea is to be able to spot the distinct trucks which would differ from the other trucks to be the 'truckMaster'.

```

Relative Position    Speed   Acceleration   Eng   Sensor Rate   Vehicle Characteristics   Behavior   Time Duration
0      14.02028  86.59919  0.56168  3.35222  ObjectDetected   Truck_Consistent  546.140277
1      16.43398  75.95607  2.37179  22.16324  ObjectDetected   Truck_Consistent  546.0335498
2      0.014825  75.95607  -9.77982  44.09591  ObjectDetected   Truck_Error        735.752685
3      52.49013  24.57039  6.62030  38.28575  ObjectDetected   Truck_Error        215.880149
4      05.85164  03.95648  0.25091  34.54108  NoObjectDetected  Truck_Consistent  203.540704

# Load the dataset from the csv file
data = pd.read_csv('truck_platoon_dataset.csv')

# Remove the vehicle characteristic column
x = data.drop(['vehicle_characteristic'], axis=1)
y = data['vehicle_characteristic']

# Encode the categorical features using LabelEncoder
label_encoder = LabelEncoder()
for column in x:
    if x[column].dtype == 'object':
        x[column] = label_encoder.fit_transform(x[column])

# Split the data into training and test datasets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

```

Fig. 10: Isolation Forest Model [1]

<https://github.com/MaximSpeczyk/Autonomouse-Lab-A-Group-4/tree/main/Leader%20vehicle%20selection/isolationForestModel>

```

# Train the isolation forest model
model = IsolationForest(n_estimators=100, max_depth=4)
model.fit(X_train)

# Predict the anomaly scores for the test dataset
anomaly_scores = model.decision_function(X_test)

# Identify the lead truck in the test dataset as the instance with the lowest anomaly score
lead_truck_index = np.argmin(anomaly_scores)
lead_truck = X_test[lead_truck_index]
test_truck = X_test[lead_truck_index]

print("Lead truck in test dataset!")
print(lead_truck)

# Calculate the anomaly score (lower score indicates a higher likelihood of being an anomaly)
# Note: g is used to classify anomalies
threshold = -anomaly_scores.mean() # You can adjust this threshold as per your requirements

# Classify samples as anomalies or not based on the threshold
predictions = (anomaly_scores <= threshold).astype(int)

# Calculate the accuracy score (percentage of correct classifications)
accuracy = (predictions == y_test).mean()
print("Accuracy: " + str(accuracy))

# Define the objective function for optimization
def accuracy_objective(threshold):
    predictions = (anomaly_scores <= threshold).astype(int)
    accuracy = (predictions == y_test).mean()
    return -accuracy # Minimize the negative accuracy to maximize accuracy

# Create a gradient function to find the threshold that achieves accuracy
result = minimize_scalar(accuracy_objective, bounds=(-100, anomaly_scores.max()), method='bounded')
best_accuracy = -result.fun

# Classify samples as anomalies or not based on the best threshold
predictions = (anomaly_scores <= best_threshold).astype(int)

# Calculate the accuracy score (percentage of correct classifications)
accuracy = (predictions == y_test).mean()
print("Best Accuracy: " + str(accuracy))
print("Best Threshold: " + str(best_threshold))

```

Fig. 11: Isolation Forest Model [2]

<https://github.com/MaximSpeczyk/Autonomouse-Lab-A-Group-4/tree/main/Leader%20vehicle%20selection/isolationForestModel>

With the random selection of data for fitting the model, the prediction accuracy is a 1.0. However, limitations lie in the overfitting of the test samples, as the model is not trained to consider all possible features, the model fitting does not have any dependency on the feature set as it is trained randomly i.e., no related data to find the 'truckMaster', and the 'Gap' feature data is incorrect as the value is expected to be constant in order to meet the 'fuel efficiency' requirement. In order to mitigate some of these constraints and optimize prediction, we can use pass optimization on the data set, have some adjustments made to the feature parameters, as well as fitting the model with the feature that have correlation for identifying which is the 'truckMaster' of the platoon.

5 Truck Platoon Scenarios Simulation

Python is used as a simulation environment for implementing the various modelled scenarios in UPPAAL. The Python code provided uses the asyncio library to simulate a platoon of vehicles with a leader and several followers.

5.1 Braking Simulation

Each vehicle is represented by a class, with the leader switching between cruising and braking states based on specified time intervals, and the followers proceed. The simulation runs to completion with all vehicles operating simultaneously as shown in [12] below.

Agent	Leader	State
class Model:		
def __init__(self, num_agents=10, num_leaders=1, seed=42):		
self.num_agents = num_agents		
self.num_leaders = num_leaders		
self.agents = [Agent(i) for i in range(num_agents)]		
self.leader = Leader(0)		
self.state = "Initial"		
self.t = 0		
if self.state == "Initial":	Leader	Crediting
self.set_state("Initial")	Agent0	Following
else:	Agent1	Following
self.set_state("Initial")	Agent2	Following
self.set_state("Initial")	Leader	Breaking
self.set_state("Initial")	Agent3	Breaking
self.set_state("Initial")	Agent4	Breaking
self.set_state("Initial")	Agent5	Breaking
self.set_state("Initial")	Agent6	Breaking
self.set_state("Initial")	Agent7	Breaking
self.set_state("Initial")	Agent8	Breaking
self.set_state("Initial")	Agent9	Breaking
class Initializer:		
def __init__(self, num_leaders=1, num_agents=10, seed=42):		
self.num_leaders = num_leaders		
self.num_agents = num_agents		
self.seed = seed		
self.agents = []		
self.leader = None		
self.state = "Initial"		
self.t = 0		
if self.state == "Initial":	Leader	Crediting
self.set_state("Initial")	Agent0	Following
else:	Agent1	Following
self.set_state("Initial")	Agent2	Following
self.set_state("Initial")	Leader	Breaking
self.set_state("Initial")	Agent3	Breaking
self.set_state("Initial")	Agent4	Breaking
self.set_state("Initial")	Agent5	Breaking
self.set_state("Initial")	Agent6	Breaking
self.set_state("Initial")	Agent7	Breaking
self.set_state("Initial")	Agent8	Breaking
self.set_state("Initial")	Agent9	Breaking
class Leader:		
def __init__(self, t=0):		
self.t = t		
def set_t(self, t):		
self.t = t		
def get_t(self):		
return self.t		
def __str__(self):		
return f"Leader at time {self.t}"		
def __repr__(self):		
return f"Leader at time {self.t}"		
class Agent:		
def __init__(self, t=0):		
self.t = t		
def set_t(self, t):		
self.t = t		
def get_t(self):		
return self.t		
def __str__(self):		
return f"Agent {self.t}"		
def __repr__(self):		
return f"Agent {self.t}"		
class State:		
def __init__(self, state="Initial", t=0):		
self.state = state		
self.t = t		
def __str__(self):		
return f"State {self.state} at time {self.t}"		
def __repr__(self):		
return f"State {self.state} at time {self.t}"		
class Model:		
def __init__(self, num_agents=10, num_leaders=1, seed=42):		
self.num_agents = num_agents		
self.num_leaders = num_leaders		
self.agents = [Agent(i) for i in range(num_agents)]		
self.leader = Leader(0)		
self.state = "Initial"		
self.t = 0		
if self.state == "Initial":	Leader	Crediting
self.set_state("Initial")	Agent0	Following
else:	Agent1	Following
self.set_state("Initial")	Agent2	Following
self.set_state("Initial")	Leader	Breaking
self.set_state("Initial")	Agent3	Breaking
self.set_state("Initial")	Agent4	Breaking
self.set_state("Initial")	Agent5	Breaking
self.set_state("Initial")	Agent6	Breaking
self.set_state("Initial")	Agent7	Breaking
self.set_state("Initial")	Agent8	Breaking
self.set_state("Initial")	Agent9	Breaking
class Initializer:		
def __init__(self, num_leaders=1, num_agents=10, seed=42):		
self.num_leaders = num_leaders		
self.num_agents = num_agents		
self.seed = seed		
self.agents = []		
self.leader = None		
self.state = "Initial"		
self.t = 0		
if self.state == "Initial":	Leader	Crediting
self.set_state("Initial")	Agent0	Following
else:	Agent1	Following
self.set_state("Initial")	Agent2	Following
self.set_state("Initial")	Leader	Breaking
self.set_state("Initial")	Agent3	Breaking
self.set_state("Initial")	Agent4	Breaking
self.set_state("Initial")	Agent5	Breaking
self.set_state("Initial")	Agent6	Breaking
self.set_state("Initial")	Agent7	Breaking
self.set_state("Initial")	Agent8	Breaking
self.set_state("Initial")	Agent9	Breaking
class Leader:		
def __init__(self, t=0):		
self.t = t		
def set_t(self, t):		
self.t = t		
def get_t(self):		
return self.t		
def __str__(self):		
return f"Leader at time {self.t}"		
def __repr__(self):		
return f"Leader at time {self.t}"		
class Agent:		
def __init__(self, t=0):		
self.t = t		
def set_t(self, t):		
self.t = t		
def get_t(self):		
return self.t		
def __str__(self):		
return f"Agent {self.t}"		
def __repr__(self):		
return f"Agent {self.t}"		
class State:		
def __init__(self, state="Initial", t=0):		
self.state = state		
self.t = t		
def __str__(self):		
return f"State {self.state} at time {self.t}"		
def __repr__(self):		
return f"State {self.state} at time {self.t}"		

Fig. 12: Braking Simulation

<https://github.com/MaximSpeczyk/Autonomous-Lab-A-Group-4/blob/main/Python%20simulations/Braking.py>

5.2 Steering Simulation

The leader truck cyclically changes its state (idle, turn left, turn right) every second and stores these states in a queue. The follower mimics the leader's states, but with a delay of 2 seconds. The script uses the `asyncio` library to allow the leader and follower to operate simultaneously in a non-blocking manner, providing a basic model of a delayed vehicle platooning scenario.

Fig. 13: Steering Simulation
<https://github.com/MaximSpeczyk/Autonomous-Lab-A-Group-4/blob/main/Python%20simulations/steering.py>

5.3 Lane Change Simulation

Here, the leader changes lanes after a certain time, and the followers proceeds after a second delay. The state of each vehicle is monitored and updated using an asynchronous event loop from the `asyncio` library. After a successful lane change, the vehicles return to their initial state of following the leader, who returns to cruising mode as shown in [14] below.

```

1 import abc
2
3 class Vehicle(abc.ABC):
4     def __init__(self, name):
5         self.name = name
6         self.state = "resting"
7
8     @property
9     def name(self):
10         return self.name
11
12     @name.setter
13     def name(self, value):
14         self.name = value
15
16     @property
17     def state(self):
18         return self.state
19
20     @state.setter
21     def state(self, value):
22         self.state = value
23
24     def __str__(self):
25         return f'{self.name} is {self.state}'
26
27     def __eq__(self, other):
28         if isinstance(other, Vehicle):
29             return self.name == other.name
30         return False
31
32     def __ne__(self, other):
33         if isinstance(other, Vehicle):
34             return self.name != other.name
35         return True
36
37     def __gt__(self, other):
38         if isinstance(other, Vehicle):
39             return self.name > other.name
40         return False
41
42     def __lt__(self, other):
43         if isinstance(other, Vehicle):
44             return self.name < other.name
45         return True
46
47     def __ge__(self, other):
48         if isinstance(other, Vehicle):
49             return self.name >= other.name
50         return False
51
52     def __le__(self, other):
53         if isinstance(other, Vehicle):
54             return self.name <= other.name
55         return True
56
57     def __hash__(self):
58         return hash(self.name)
59
60     def __repr__(self):
61         return f'Vehicle({self.name})'
62
63     def __enter__(self):
64         self.state = "driving"
65         return self
66
67     def __exit__(self, exc_type, exc_value, exc_traceback):
68         self.state = "resting"
69
70     def __bool__(self):
71         return self.state == "driving"
72
73     def __nonzero__(self):
74         return self.state == "driving"
75
76     def __len__(self):
77         return len(self.name)
78
79     def __iter__(self):
80         for character in self.name:
81             yield character
82
83     def __next__(self):
84         if self.state == "resting":
85             raise StopIteration()
86         else:
87             self.state = "resting"
88             return next(self)
89
90     def __reversed__(self):
91         if self.state == "resting":
92             raise StopIteration()
93         else:
94             self.state = "resting"
95             return reversed(self)
96
97     def __format__(self, format_spec):
98         if self.state == "resting":
99             return f'{self.name} is resting'
100        else:
101            return f'{self.name} is driving'
102
103    def __getattribute__(self, name):
104        if name == "state" and self.state == "resting":
105            return "driving"
106        return super().__getattribute__(name)
107
108    def __setattr__(self, name, value):
109        if name == "state" and value == "driving" and self.state == "resting":
110            self.state = "resting"
111        super().__setattr__(name, value)
112
113    def __delattr__(self, name):
114        if name == "state" and value == "driving" and self.state == "resting":
115            self.state = "resting"
116        super().__delattr__(name)
117
118    def __call__(self, *args, **kwargs):
119        print(f'{self.name} is calling')
120
121    def __del__(self):
122        print(f'{self.name} is deleted')
123
124    def __copy__(self):
125        return Vehicle(self.name)
126
127    def __deepcopy__(self, memo):
128        return Vehicle(self.name)
129
130    def __new__(cls, name):
131        return cls(name)
132
133    def __reduce__(self):
134        return (Vehicle, (self.name,))
135
136    def __reduce_ex__(self, protocol):
137        return (Vehicle, (self.name,))
138
139    def __getstate__(self):
140        return self.name
141
142    def __setstate__(self, name):
143        self.name = name
144
145    def __getpicklable__(self):
146        return self.name
147
148    def __setpicklable__(self, name):
149        self.name = name
150
151    def __getstate__(self):
152        return self.name
153
154    def __setstate__(self, name):
155        self.name = name
156
157    def __getpicklable__(self):
158        return self.name
159
160    def __setpicklable__(self, name):
161        self.name = name
162
163    def __getstate__(self):
164        return self.name
165
166    def __setstate__(self, name):
167        self.name = name
168
169    def __getpicklable__(self):
170        return self.name
171
172    def __setpicklable__(self, name):
173        self.name = name
174
175    def __getstate__(self):
176        return self.name
177
178    def __setstate__(self, name):
179        self.name = name
180
181    def __getpicklable__(self):
182        return self.name
183
184    def __setpicklable__(self, name):
185        self.name = name
186
187    def __getstate__(self):
188        return self.name
189
190    def __setstate__(self, name):
191        self.name = name
192
193    def __getpicklable__(self):
194        return self.name
195
196    def __setpicklable__(self, name):
197        self.name = name
198
199    def __getstate__(self):
200        return self.name
201
202    def __setstate__(self, name):
203        self.name = name
204
205    def __getpicklable__(self):
206        return self.name
207
208    def __setpicklable__(self, name):
209        self.name = name
210
211    def __getstate__(self):
212        return self.name
213
214    def __setstate__(self, name):
215        self.name = name
216
217    def __getpicklable__(self):
218        return self.name
219
220    def __setpicklable__(self, name):
221        self.name = name
222
223    def __getstate__(self):
224        return self.name
225
226    def __setstate__(self, name):
227        self.name = name
228
229    def __getpicklable__(self):
230        return self.name
231
232    def __setpicklable__(self, name):
233        self.name = name
234
235    def __getstate__(self):
236        return self.name
237
238    def __setstate__(self, name):
239        self.name = name
240
241    def __getpicklable__(self):
242        return self.name
243
244    def __setpicklable__(self, name):
245        self.name = name
246
247    def __getstate__(self):
248        return self.name
249
250    def __setstate__(self, name):
251        self.name = name
252
253    def __getpicklable__(self):
254        return self.name
255
256    def __setpicklable__(self, name):
257        self.name = name
258
259    def __getstate__(self):
260        return self.name
261
262    def __setstate__(self, name):
263        self.name = name
264
265    def __getpicklable__(self):
266        return self.name
267
268    def __setpicklable__(self, name):
269        self.name = name
270
271    def __getstate__(self):
272        return self.name
273
274    def __setstate__(self, name):
275        self.name = name
276
277    def __getpicklable__(self):
278        return self.name
279
280    def __setpicklable__(self, name):
281        self.name = name
282
283    def __getstate__(self):
284        return self.name
285
286    def __setstate__(self, name):
287        self.name = name
288
289    def __getpicklable__(self):
290        return self.name
291
292    def __setpicklable__(self, name):
293        self.name = name
294
295    def __getstate__(self):
296        return self.name
297
298    def __setstate__(self, name):
299        self.name = name
300
301    def __getpicklable__(self):
302        return self.name
303
304    def __setpicklable__(self, name):
305        self.name = name
306
307    def __getstate__(self):
308        return self.name
309
310    def __setstate__(self, name):
311        self.name = name
312
313    def __getpicklable__(self):
314        return self.name
315
316    def __setpicklable__(self, name):
317        self.name = name
318
319    def __getstate__(self):
320        return self.name
321
322    def __setstate__(self, name):
323        self.name = name
324
325    def __getpicklable__(self):
326        return self.name
327
328    def __setpicklable__(self, name):
329        self.name = name
330
331    def __getstate__(self):
332        return self.name
333
334    def __setstate__(self, name):
335        self.name = name
336
337    def __getpicklable__(self):
338        return self.name
339
340    def __setpicklable__(self, name):
341        self.name = name
342
343    def __getstate__(self):
344        return self.name
345
346    def __setstate__(self, name):
347        self.name = name
348
349    def __getpicklable__(self):
350        return self.name
351
352    def __setpicklable__(self, name):
353        self.name = name
354
355    def __getstate__(self):
356        return self.name
357
358    def __setstate__(self, name):
359        self.name = name
360
361    def __getpicklable__(self):
362        return self.name
363
364    def __setpicklable__(self, name):
365        self.name = name
366
367    def __getstate__(self):
368        return self.name
369
370    def __setstate__(self, name):
371        self.name = name
372
373    def __getpicklable__(self):
374        return self.name
375
376    def __setpicklable__(self, name):
377        self.name = name
378
379    def __getstate__(self):
380        return self.name
381
382    def __setstate__(self, name):
383        self.name = name
384
385    def __getpicklable__(self):
386        return self.name
387
388    def __setpicklable__(self, name):
389        self.name = name
390
391    def __getstate__(self):
392        return self.name
393
394    def __setstate__(self, name):
395        self.name = name
396
397    def __getpicklable__(self):
398        return self.name
399
400    def __setpicklable__(self, name):
401        self.name = name
402
403    def __getstate__(self):
404        return self.name
405
406    def __setstate__(self, name):
407        self.name = name
408
409    def __getpicklable__(self):
410        return self.name
411
412    def __setpicklable__(self, name):
413        self.name = name
414
415    def __getstate__(self):
416        return self.name
417
418    def __setstate__(self, name):
419        self.name = name
420
421    def __getpicklable__(self):
422        return self.name
423
424    def __setpicklable__(self, name):
425        self.name = name
426
427    def __getstate__(self):
428        return self.name
429
430    def __setstate__(self, name):
431        self.name = name
432
433    def __getpicklable__(self):
434        return self.name
435
436    def __setpicklable__(self, name):
437        self.name = name
438
439    def __getstate__(self):
440        return self.name
441
442    def __setstate__(self, name):
443        self.name = name
444
445    def __getpicklable__(self):
446        return self.name
447
448    def __setpicklable__(self, name):
449        self.name = name
450
451    def __getstate__(self):
452        return self.name
453
454    def __setstate__(self, name):
455        self.name = name
456
457    def __getpicklable__(self):
458        return self.name
459
460    def __setpicklable__(self, name):
461        self.name = name
462
463    def __getstate__(self):
464        return self.name
465
466    def __setstate__(self, name):
467        self.name = name
468
469    def __getpicklable__(self):
470        return self.name
471
472    def __setpicklable__(self, name):
473        self.name = name
474
475    def __getstate__(self):
476        return self.name
477
478    def __setstate__(self, name):
479        self.name = name
480
481    def __getpicklable__(self):
482        return self.name
483
484    def __setpicklable__(self, name):
485        self.name = name
486
487    def __getstate__(self):
488        return self.name
489
490    def __setstate__(self, name):
491        self.name = name
492
493    def __getpicklable__(self):
494        return self.name
495
496    def __setpicklable__(self, name):
497        self.name = name
498
499    def __getstate__(self):
500        return self.name
501
502    def __setstate__(self, name):
503        self.name = name
504
505    def __getpicklable__(self):
506        return self.name
507
508    def __setpicklable__(self, name):
509        self.name = name
510
511    def __getstate__(self):
512        return self.name
513
514    def __setstate__(self, name):
515        self.name = name
516
517    def __getpicklable__(self):
518        return self.name
519
520    def __setpicklable__(self, name):
521        self.name = name
522
523    def __getstate__(self):
524        return self.name
525
526    def __setstate__(self, name):
527        self.name = name
528
529    def __getpicklable__(self):
530        return self.name
531
532    def __setpicklable__(self, name):
533        self.name = name
534
535    def __getstate__(self):
536        return self.name
537
538    def __setstate__(self, name):
539        self.name = name
540
541    def __getpicklable__(self):
542        return self.name
543
544    def __setpicklable__(self, name):
545        self.name = name
546
547    def __getstate__(self):
548        return self.name
549
550    def __setstate__(self, name):
551        self.name = name
552
553    def __getpicklable__(self):
554        return self.name
555
556    def __setpicklable__(self, name):
557        self.name = name
558
559    def __getstate__(self):
560        return self.name
561
562    def __setstate__(self, name):
563        self.name = name
564
565    def __getpicklable__(self):
566        return self.name
567
568    def __setpicklable__(self, name):
569        self.name = name
570
571    def __getstate__(self):
572        return self.name
573
574    def __setstate__(self, name):
575        self.name = name
576
577    def __getpicklable__(self):
578        return self.name
579
580    def __setpicklable__(self, name):
581        self.name = name
582
583    def __getstate__(self):
584        return self.name
585
586    def __setstate__(self, name):
587        self.name = name
588
589    def __getpicklable__(self):
590        return self.name
591
592    def __setpicklable__(self, name):
593        self.name = name
594
595    def __getstate__(self):
596        return self.name
597
598    def __setstate__(self, name):
599        self.name = name
600
601    def __getpicklable__(self):
602        return self.name
603
604    def __setpicklable__(self, name):
605        self.name = name
606
607    def __getstate__(self):
608        return self.name
609
610    def __setstate__(self, name):
611        self.name = name
612
613    def __getpicklable__(self):
614        return self.name
615
616    def __setpicklable__(self, name):
617        self.name = name
618
619    def __getstate__(self):
620        return self.name
621
622    def __setstate__(self, name):
623        self.name = name
624
625    def __getpicklable__(self):
626        return self.name
627
628    def __setpicklable__(self, name):
629        self.name = name
630
631    def __getstate__(self):
632        return self.name
633
634    def __setstate__(self, name):
635        self.name = name
636
637    def __getpicklable__(self):
638        return self.name
639
640    def __setpicklable__(self, name):
641        self.name = name
642
643    def __getstate__(self):
644        return self.name
645
646    def __setstate__(self, name):
647        self.name = name
648
649    def __getpicklable__(self):
650        return self.name
651
652    def __setpicklable__(self, name):
653        self.name = name
654
655    def __getstate__(self):
656        return self.name
657
658    def __setstate__(self, name):
659        self.name = name
660
661    def __getpicklable__(self):
662        return self.name
663
664    def __setpicklable__(self, name):
665        self.name = name
666
667    def __getstate__(self):
668        return self.name
669
670    def __setstate__(self, name):
671        self.name = name
672
673    def __getpicklable__(self):
674        return self.name
675
676    def __setpicklable__(self, name):
677        self.name = name
678
679    def __getstate__(self):
680        return self.name
681
682    def __setstate__(self, name):
683        self.name = name
684
685    def __getpicklable__(self):
686        return self.name
687
688    def __setpicklable__(self, name):
689        self.name = name
690
691    def __getstate__(self):
692        return self.name
693
694    def __setstate__(self, name):
695        self.name = name
696
697    def __getpicklable__(self):
698        return self.name
699
700    def __setpicklable__(self, name):
701        self.name = name
702
703    def __getstate__(self):
704        return self.name
705
706    def __setstate__(self, name):
707        self.name = name
708
709    def __getpicklable__(self):
710        return self.name
711
712    def __setpicklable__(self, name):
713        self.name = name
714
715    def __getstate__(self):
716        return self.name
717
718    def __setstate__(self, name):
719        self.name = name
720
721    def __getpicklable__(self):
722        return self.name
723
724    def __setpicklable__(self, name):
725        self.name = name
726
727    def __getstate__(self):
728        return self.name
729
730    def __setstate__(self, name):
731        self.name = name
732
733    def __getpicklable__(self):
734        return self.name
735
736    def __setpicklable__(self, name):
737        self.name = name
738
739    def __getstate__(self):
740        return self.name
741
742    def __setstate__(self, name):
743        self.name = name
744
745    def __getpicklable__(self):
746        return self.name
747
748    def __setpicklable__(self, name):
749        self.name = name
750
751    def __getstate__(self):
752        return self.name
753
754    def __setstate__(self, name):
755        self.name = name
756
757    def __getpicklable__(self):
758        return self.name
759
760    def __setpicklable__(self, name):
761        self.name = name
762
763    def __getstate__(self):
764        return self.name
765
766    def __setstate__(self, name):
767        self.name = name
768
769    def __getpicklable__(self):
770        return self.name
771
772    def __setpicklable__(self, name):
773        self.name = name
774
775    def __getstate__(self):
776        return self.name
777
778    def __setstate__(self, name):
779        self.name = name
780
781    def __getpicklable__(self):
782        return self.name
783
784    def __setpicklable__(self, name):
785        self.name = name
786
787    def __getstate__(self):
788        return self.name
789
790    def __setstate__(self, name):
791        self.name = name
792
793    def __getpicklable__(self):
794        return self.name
795
796    def __setpicklable__(self, name):
797        self.name = name
798
799    def __getstate__(self):
800        return self.name
801
802    def __setstate__(self, name):
803        self.name = name
804
805    def __getpicklable__(self):
806        return self.name
807
808    def __setpicklable__(self, name):
809        self.name = name
810
811    def __getstate__(self):
812        return self.name
813
814    def __setstate__(self, name):
815        self.name = name
816
817    def __getpicklable__(self):
818        return self.name
819
820    def __setpicklable__(self, name):
821        self.name = name
822
823    def __getstate__(self):
824        return self.name
825
826    def __setstate__(self, name):
827        self.name = name
828
829    def __getpicklable__(self):
830        return self.name
831
832    def __setpicklable__(self, name):
833        self.name = name
834
835    def __getstate__(self):
836        return self.name
837
838    def __setstate__(self, name):
839        self.name = name
840
841    def __getpicklable__(self):
842        return self.name
843
844    def __setpicklable__(self, name):
845        self.name = name
846
847    def __getstate__(self):
848        return self.name
849
850    def __setstate__(self, name):
851        self.name = name
852
853    def __getpicklable__(self):
854        return self.name
855
856    def __setpicklable__(self, name):
857        self.name = name
858
859    def __getstate__(self):
860        return self.name
861
862    def __setstate__(self, name):
863        self.name = name
864
865    def __getpicklable__(self):
866        return self.name
867
868    def __setpicklable__(self, name):
869        self.name = name
870
871    def __getstate__(self):
872        return self.name
873
874    def __setstate__(self, name):
875        self.name = name
876
877    def __getpicklable__(self):
878        return self.name
879
880    def __setpicklable__(self, name):
881        self.name = name
882
883    def __getstate__(self):
884        return self.name
885
886    def __setstate__(self, name):
887        self.name = name
888
889    def __getpicklable__(self):
890        return self.name
891
892    def __setpicklable__(self, name):
893        self.name = name
894
895    def __getstate__(self):
896        return self.name
897
898    def __setstate__(self, name):
899        self.name = name
900
901    def __getpicklable__(self):
902        return self.name
903
904    def __setpicklable__(self, name):
905        self.name = name
906
907    def __getstate__(self):
908        return self.name
909
910    def __setstate__(self, name):
911        self.name = name
912
913    def __getpicklable__(self):
914        return self.name
915
916    def __setpicklable__(self, name):
917        self.name = name
918
919    def __getstate__(self):
920        return self.name
921
922    def __setstate__(self, name):
923        self.name = name
924
925    def __getpicklable__(self):
926        return self.name
927
928    def __setpicklable__(self, name):
929        self.name = name
930
931    def __getstate__(self):
932        return self.name
933
934    def __setstate__(self, name):
935        self.name = name
936
937    def __getpicklable__(self):
938        return self.name
939
940    def __setpicklable__(self, name):
941        self.name = name
942
943    def __getstate__(self):
944        return self.name
945
946    def __setstate__(self, name):
947        self.name = name
948
949    def __getpicklable__(self):
950        return self.name
951
952    def __setpicklable__(self, name):
953        self.name = name
954
955    def __getstate__(self):
956        return self.name
957
958    def __setstate__(self, name):
959        self.name = name
960
961    def __getpicklable__(self):
962        return self.name
963
964    def __setpicklable__(self, name):
965        self.name = name
966
967    def __getstate__(self):
968        return self.name
969
970    def __setstate__(self, name):
971        self.name = name
972
973    def __getpicklable__(self):
974        return self.name
975
976    def __setpicklable__(self, name):
977        self.name = name
978
979    def __getstate__(self):
980        return self.name
981
982    def __setstate__(self, name):
983        self.name = name
984
985    def __getpicklable__(self):
986        return self.name
987
988    def __setpicklable__(self, name):
989        self.name = name
990
991    def __getstate__(self):
992        return self.name
993
994    def __setstate__(self, name):
995        self.name = name
996
997    def __getpicklable__(self):
998        return self.name
999
1000    def __setpicklable__(self, name):
1001        self.name = name
1002
1003    def __getstate__(self):
1004        return self.name
1005
1006    def __setstate__(self, name):
1007        self.name = name
1008
1009    def __getpicklable__(self):
1010        return self.name
1011
1012    def __setpicklable__(self, name):
1013        self.name = name
1014
1015    def __getstate__(self):
1016        return self.name
1017
1018    def __setstate__(self, name):
1019        self.name = name
1020
1021    def __getpicklable__(self):
1022        return self.name
1023
1024    def __setpicklable__(self, name):
1025        self.name = name
1026
1027    def __getstate__(self):
1028        return self.name
1029
1030    def __setstate__(self, name):
1031        self.name = name
1032
1033    def __getpicklable__(self):
1034        return self.name
1035
1036    def __setpicklable__(self, name):
1037        self.name = name
1038
1039    def __getstate__(self):
1040        return self.name
1041
1042    def __setstate__(self, name):
1043        self.name = name
1044
1045    def __getpicklable__(self):
1046        return self.name
1047
1048    def __setpicklable__(self, name):
1049        self.name = name
1050
1051    def __getstate__(self):
1052        return self.name
1053
1054    def __setstate__(self, name):
1055        self.name = name
1056
1057    def __getpicklable__(self):
1058        return self.name
1059
1060    def __setpicklable__(self, name):
1061        self.name = name
1062
1063    def __getstate__(self):
1064        return self.name
1065
1066    def __setstate__(self, name):
1067        self.name = name
1068
1069    def __getpicklable__(self):
1070        return self.name
1071
1072    def __setpicklable__(self, name):
1073        self.name = name
1074
1075    def __getstate__(self):
1076        return self.name
1077
1078    def __setstate__(self, name):
1079        self.name = name
1080
1081    def __getpicklable__(self):
1082        return self.name
1083
1084    def __setpicklable__(self, name):
1085        self.name = name
1086
1087    def __getstate__(self):
1088        return self.name
1089
1090    def __setstate__(self, name):
1091        self.name = name
1092
1093    def __getpicklable__(self):
1094        return self.name
1095
1096    def __setpicklable__(self, name):
1097        self.name = name
1098
1099    def __getstate__(self):
1100        return self.name
1101
1102    def __setstate__(self, name):
1103        self.name = name
1104
1105    def __getpicklable__(self):
1106        return self.name
1107
1108    def __setpicklable__(self, name):
1109        self.name = name
1110
1111    def __getstate__(self):
1112        return self.name
1113
1114    def __setstate__(self, name):
1115        self.name = name
1116
1117    def __getpicklable__(self):
1118        return self.name
1119
1120    def __setpicklable__(self, name):
1121        self.name = name
1122
1123    def __getstate__(self):
1124        return self.name
1125
1126    def __setstate__(self, name):
1127        self.name = name
1128
1129    def __getpicklable__(self):
1130        return self.name
1131
1132    def __setpicklable__(self, name):
1133        self.name = name
1134
1135    def __getstate__(self):
1136        return self.name
1137
1138    def __setstate__(self, name):
1139        self.name = name
1140
1141    def __getpicklable__(self):
1142        return self.name
1143
1144    def __setpicklable__(self, name):
1145        self.name = name
1146
1147    def __getstate__(self):
1148        return self.name
1149
1150    def __setstate__(self, name):
1151        self.name = name
1152
1153    def __getpicklable__(self):
1154        return self.name
1155
1156    def __setpicklable__(self, name):
1157        self.name = name
1158
1159    def __getstate__(self):
1160        return self.name
1161
1162    def __setstate__(self, name):
1163        self.name = name
1164
1165    def __getpicklable__(self):
1166        return self.name
1167
1168    def __setpicklable__(self, name):
1169        self.name = name
1170
1171    def __getstate__(self):
1172        return self.name
1173
1174    def __setstate__(self, name):
1175        self.name = name
1176
1177    def __getpicklable__(self):
1178        return self.name
1179
1180    def __setpicklable__(self, name):
1181        self.name = name
1182
1183    def __getstate__(self):
1184        return self.name
1185
1186    def __setstate__(self, name):
1187        self.name = name
1188
1189    def __getpicklable__(self):
1190        return self.name
1191
1192    def __setpicklable__(self, name):
1193        self.name = name
1194
1195    def __getstate__(self):
1196        return self.name
1197
1198    def __setstate__(self, name):
1199        self.name = name
1200
1201    def __getpicklable__(self):
1202        return self.name
1203
1204    def __setpicklable__(self, name):
1205        self.name = name
1206
1207    def __getstate__(self):
1208        return self.name
1209
1210    def __setstate__(self, name):
1211        self.name = name
1212
1213    def __getpicklable__(self):
1214        return self.name
1215
1216    def __setpicklable__(self, name):
1217        self.name = name
1218
1219    def __getstate__(self):
1220        return self.name
1221
1222    def __setstate__(self, name):
1223        self.name = name
1224
1225    def __getpicklable__(self):
1226        return self.name
1227
1228    def __setpicklable__(self, name):
1229        self.name = name
1230
1231    def __getstate__(self):
1232        return self.name
1233
1234    def __setstate__(self, name):
1235        self.name = name
1236
1237    def __getpicklable__(self):
1238        return self.name
1239
1240    def __setpicklable__(self, name):
1241        self.name = name
1242
1243    def __getstate__(self):
1244        return self.name
1245
1246    def __setstate__(self, name):
1247        self.name = name
1248
1249    def __getpicklable__(self):
1250        return self.name
1251
1252    def __setpicklable__(self, name):
1253        self.name = name
1254
1255    def __getstate__(self):
1256        return self.name
1257
1258    def __setstate__(self, name):
1259        self.name = name
1260
1261    def __getpicklable__(self):
1262        return self.name
1263
1264    def __setpicklable__(self, name):
1265        self.name = name
1266
1267    def __getstate__(self):
1268        return self.name
1269
1270    def __setstate__(self, name):
1271        self.name = name
1272
1273    def __getpickl
```

Date & Place - Muhammad Umer Bin Yaqoob