Truck Platoon

[1]

Contents

[1]

Abstract: This documentation describes an innovative method to autonomous lab truck platooning that focuses on several scenerios that essential parts of platooning. We used a variety of advanced approaches to achieve these aims, including diagrams to explain the scenerios and the project, timed automata modeled with Uppaal, machine learning algorithm based on decision trees, and simulation using the Pygame framework.

Keywords: truck platooning, timed automata, decision trees, simulation

1 Introduction

Truck platoon is a fascinating topic that is being worked on by some truck manufacturing companies such as Volkswagen. Truck platooning is a solution for truck traffic on highways. Making trucks travel in a highway lane with a specific amount of space between them is a pretty simple approach for ensuring safe clearance. The vehicles communicate with one another and travel at the same speed. As part of this coordination, vehicles exchange data and messages. Some advantages of truck platooning includes; smooth traffic flow, good fuel economy and less human intervention. In Autonomous A lab we got the opportunity to work on this topic, study its requirements, scenarios and finally implement them. There was a great deal of learning for us in this project.

2 Task 1

2.1 Requirement Diagram (Mohammad Ashraf Siddiqi)

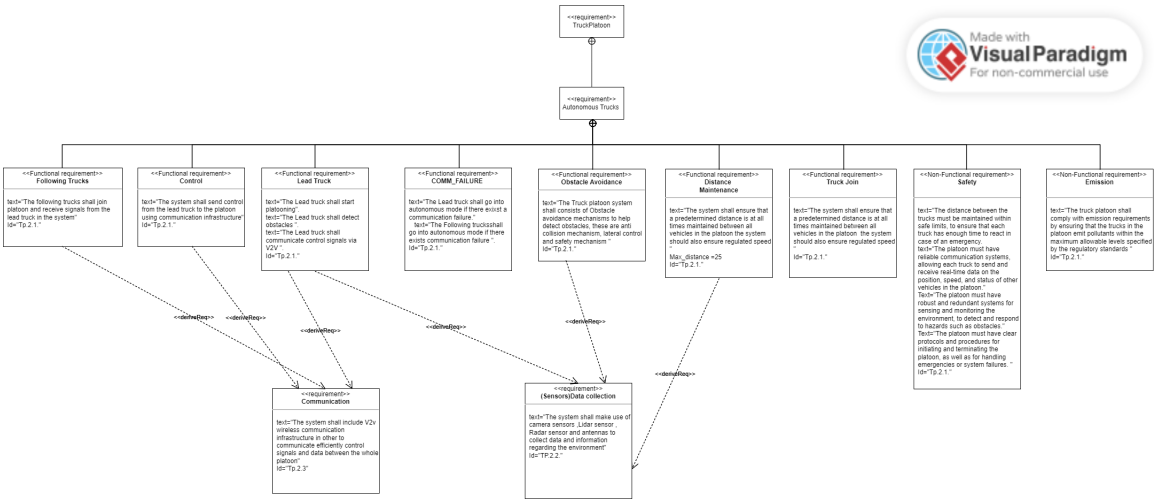


Fig. 1: Requirement Truck Platoon

2.2 System requirements and Specific scenarios (Team work)

- Communication: The trucks in the platoon communicate with one another, exchanging information such as speed, position, and acceleration. This communication allows for coordinated movement and aids in the maintenance of a synchronized formation.
- Distance Maintenance: The trucks in the platoon keep a set distance between themselves, which is typically referred to as the "max-distín our uppaal models. This

distance is carefully managed to ensure safe and efficient operation, limiting collision risk and maximizing aerodynamic benefits.

- Truck Joining: When a new truck wants to join the platoon, it aligns itself with the lead truck and adjusts its speed to match the platoon's speed. By following the lead truck, the joining truck becomes part of the platoon, benefiting from improved fuel efficiency.
- Sensors: Each truck in the platoon is equipped with various sensors, such as radar or LiDAR, to detect obstacles and monitor the surrounding environment. These sensors provide real-time data to the truck's control system, enabling it to make informed decisions regarding acceleration, braking, and steering.
- Handling Communication Failures: In the event of a communication failure between the trucks, such as a lost signal or a disturbed connection, the trucks are programmed to go into autonomous mode. Each vehicle then employs its internal sensors and control algorithms to keep a safe distance, avoid obstructions, and operate autonomously until contact is restored.
- Obstacle Avoidance: A platoon's trucks are outfitted with advanced obstacle detection and avoidance technologies, such as radar and camera-based sensors. These technologies constantly watch the road ahead, detect possible hazards such as vehicles or people, and offer real-time data to the truck's control system. The trucks can prevent crashes and ensure safe passage by proactively adjusting their speed or changing lanes based on this data.
- Safety: Technologies and practices ensure truck platoon safety. Synchronized movement and regular communication between trucks reduce accidents caused by unexpected lane changes or braking. Advanced safety features including collision warning systems and automatic emergency braking safeguard platoon members and other road users.
- Truck platooning reduces pollutants and optimizes fuel efficiency. Aerodynamics and fuel economy are improved by platoon trucks' close following distances. This fuel efficiency enhancement reduces emissions, making transportation greener. Trucking may reduce its carbon impact and promote green practices by using platooning technology.

2.3 Specific scenarios

2.3.1 Joining Truck Scenario (Doluwamu Taiwo Kuye)

The Joining vehicle commences the sequence by requesting to join the platoon; the lead truck acknowledges the request and updates the platoon's current following trucks. Following this, the essential data for joining is provided to the joining truck, which adjusts its own parameters (speed and distance) in order to connect; the system then guarantees that every member is updated with the latest data.

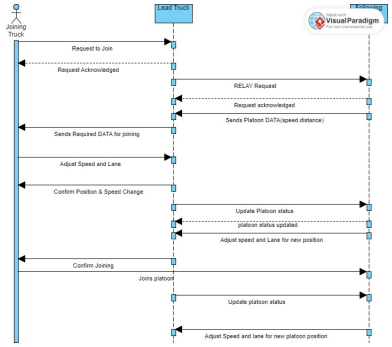


Fig. 2: Joining truck Scenario

2.3.2 Obstacle avoidance (Arsal Abbasi & Doluwamu Taiwo Kuye)

If the sensor detects an obstruction, the lead truck alerts the following trucks, and if the following vehicles acknowledge this, the lead truck adjusts the speed and position and performs obstacle avoidance. If this is completed, an update is provided to the overall system.

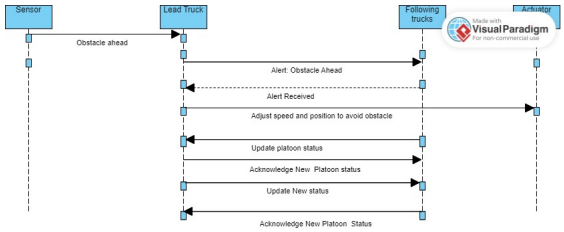


Fig. 3: Obstacle Avoidance

2.3.3 Maintaining Distance (Arsal Abbasi)

In our maintaining distance scenerio, the driver of the lead truck first makes assessment of the conditions and depending upon that he makes the decision of starting platoon. After initiating platoon following trucks join the platoon by requesting for it. Our lead truck consists of sensors that monitor the conditions for distance maintanance. According to the monitored data, controller in lead truck makes the decision to actuate the actuators present in the following trucks, wither increase or decrease the distance. Figure 4 represents the sequence diagram of our maintaining distance scenerio.

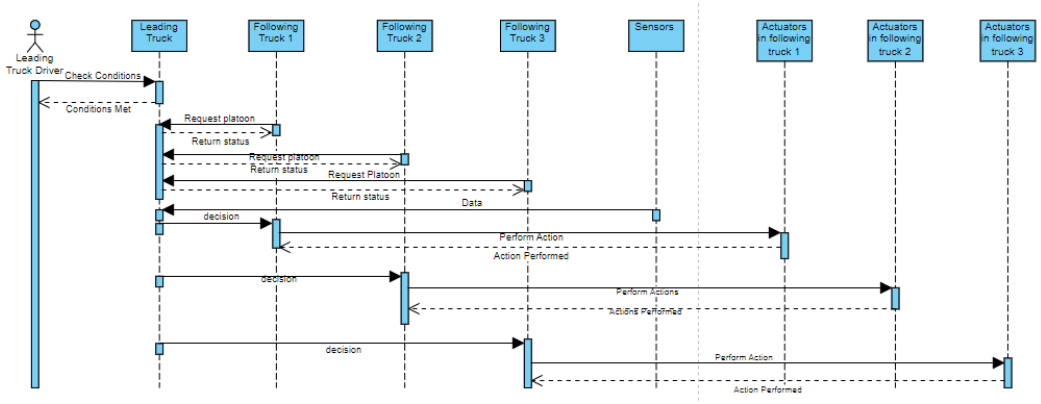


Fig. 4: Maintaining Distance

3 Task 2 and 3

In this Task, we integrated the timed automata from task 2 and added some exclusive scenarios like comm-failure and Truck Joining before the platoon starts and after.

3.1 Lead Truck Automata (Arsal Abbasi & Doluwamu Taiwo kuye)

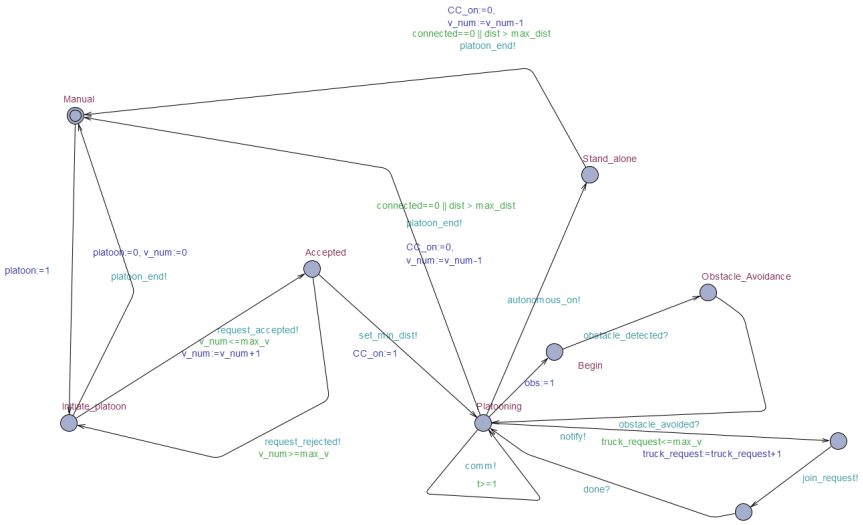


Fig. 5: Lead

This model describes the behavior of our lead truck and all the possible states that it can transition into. We highlight the control behavior of our system using this model.

1. Initiate platoon: After starting the platoon by updating the platoon variable to 1 we move to the initiate platoon state. In this state we send the accept request sync after checking the guard (vehicle number<=max vehicles), this is to ensure that the number of vehicles accepted to join the platoon is within the requirements of our truck.
2. Accepted State: In this state, we set the minimum distance for the platooning to ensure maintaining distance scenario and also update the cruise control to 1 to ensure that all members of the platoon maintain the same cruise speed. These 2 mechanisms ensure maintaining distance in our platoon.
3. Platooning: The platooning state is a major component of the system as it allows 5 different transitions, such as
   - Comm: to show continuous communication with the following trucks

- TruckJoin: In this state, we either accept or deny the join request and continue platooning after
- Obstacle Avoidance: on obs being set to 1 we synchronize with the obstacle avoidance automata and commence the obstacle avoidance procedure then return to the platooning state.
- Autonomous-on (comm-Failure): In case of autonomous failure in the lead truck we transition to the stand-alone state with sets on our lead controller automata.

4. Standalone: In stand-alone mode, our lead truck is independent of the platoon, and is controlled by another-automata in 5.

3.2 Follow Truck Automata (Arsal Abbasi & Doluwamu Taiwo kuye)

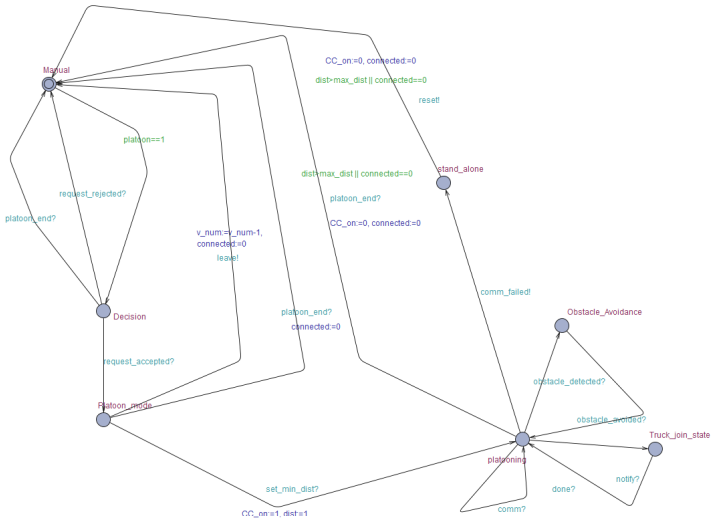


Fig. 6: Follow truck

The following truck automata specify the interactions between the lead and following, most of the states are synchronized with the lead truck and depend on signals from the lead truck to transition. It also includes a standalone state in case of communication failure or a break with the lead truck, this allows it to control itself.

3.3 Truck Join Automata (Arsal Abbasi & Doluwamu Taiwo Kuye)

In this automata shown in figure 10 placed in the Appendix A we specify all the necessary transitions for the truck joining process, it is important to know that the input for this system is received from a component automata shown in figure 9, this serves as a middleman to ensure synchronization of the system.

3.4 Obstacle Avoidance Automata (Doluwamu Taiwo kuye)

This is activated anytime the (obs) variable is updated to 1 in the lead truck, it shows that the sensor has picked up an obstacle. A guard in the obstacle automata checks if the obs is 1 if this is satisfied it begins the obstacle avoidance sequence. The automata is shown in figure 13 placed in Appendix A.

3.5 Control Automata (Arsal Abbasi)

We have implemented 2 automata for We implemented two automata for controlling the lead truck and following trucks in case of communication failure. They also ensure the safe distance between the now separated trucks in the platoon is maintained. Hence the distance is maintained even on the case of communication failure. Automata are shown in figure 12 and 11 both placed in the Appendix A,

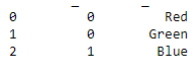
4 Task 4 (Arsal Abbasi)

In this task, we implemented a machine learning algorithm using SciKit learn. Having a technology that can make decisions when required without human interference is very crucial for our system. We used machine learning algorithm in order to determine lead truck among three platooning trucks. The algorithm we decided for this purpose is decision tree. We used Scikit learn library in Python to implement decision tree algorithm. Decision trees is a supervised learning technique that has been widely used to develop classification models because they closely mirror human thinking and are simple to grasp [**?**]. Decision tree algorithm is created by training the model. It gives the predicted final value based on training data. To get the data that can train our model perfectly so that our model always predicts the correct lead truck based on the characteristics was quite a challenging part. In our project, the decision of the lead truck should be based on the characteristics; truck height, truck width, distance of the route, max match of the route, fuel, number of sensors and speed. The truck which has most of these characteristics more than the other trucks will be decided as the lead truck. In our data set we train the model with truck ID. In training data when the truck has most of these characteristics it is given truck id 1, which makes it the lead truck. The data set was created manually, as random generator sometimes gave us results that were not reliable. Figure 8 represents the data. This data was split into two, 80% for training and 20% for testing.

When our model gets characteristics of the new trucks, it predicts the lead truck. For our project, the new data we gave our model is:

```
new_data = {
    'truck_color': ['Red', 'Green', 'Blue'],
    'truck_height': [10, 19, 20],
    'truck_width': [17, 18, 20],
    'distance_of_route':[200, 250, 260],
    'max_match_of_route':[200, 250, 250],
    'fuel':[25, 25, 25],
    'number_of_sensors': [22, 23, 24],
    'speed': [125, 135, 140]
}
```

Based on this new data one can guess that the lead truck should Blue truck as it has most of the characteristics, and our model guessed the same. It predicted blue because of it has most characteristics, the same way if red or green had more then they would have been predicted as a lead truck. Figure 7 represents the results of the new data, in which lead truck is predicted to be the blue one, with the accuracy of 66%.



```
   _   _
0  0   Red
1  0  Green
2  1   Blue
```

Fig. 7: New data prediction using implemented decision tree

	truck_id	truck_color	truck_height	truck_width	distance_of_route	max_match_of_route	fuel	number_of_sensors	speed
0	1	Red	15	10	250	200	25	7	80
1	2	Green	10	8	150	150	15	6	75
2	3	Blue	12	9	200	50	20	5	78
3	1	Green	15	10	200	150	30	12	90
4	2	Red	8	8	100	100	10	7	85
5	3	Blue	8	10	150	150	15	10	82
6	1	Blue	12	12	150	150	20	16	89
7	2	Green	11	11	100	100	10	7	72
8	3	Red	9	11	50	50	5	15	88
9	1	Red	13	10	300	200	30	10	92
10	2	Green	10	9	100	100	10	6	86
11	3	Blue	9	10	50	50	5	8	80
12	1	Blue	12	12	250	150	25	15	90
13	2	Green	9	10	200	50	10	11	82
14	3	Red	10	11	200	100	5	10	80
15	1	Red	15	10	250	200	25	7	80
16	2	Green	10	8	150	150	15	6	75
17	3	Blue	12	9	200	50	20	5	78
18	1	Green	15	10	200	150	30	12	90
19	2	Red	8	8	100	100	10	7	85
20	3	Blue	8	10	150	150	15	10	82
21	1	Blue	12	12	150	150	20	16	89
22	2	Green	11	11	100	100	10	7	72
23	3	Red	9	11	50	50	5	15	88
24	1	Red	13	10	300	200	30	10	92
25	2	Green	10	9	100	100	10	6	86
26	3	Blue	9	10	50	50	5	8	80
27	1	Blue	12	12	250	150	25	15	90
28	2	Green	9	10	200	50	10	11	82
29	3	Red	10	11	200	100	5	10	80

Fig. 8: Training data-set

Complete machine learning code can be viewed here (Click Here)

5 Task 5(Simulation)

For the simulation, we integrated the decision tree algorithm in the pygame environment where we simulated joining truck, maintaining distance and obstacle avoidance.

5.1 Maintaining distance (Arsal Abbasi)

Machine Learning: The code uses the DecisionTreeClassifier from the scikit-learn library (sklearn.tree) for the machine learning part can be seen in the appendix A in figure 14.

- Integration with Maintaining Distance:
  - The code sets up a simulation environment where trucks maintain a certain distance from each other.
  - The trucks list contains dictionaries representing the trucks in the simulation. Each dictionary includes attributes such as id, x, y, dx, dy, color, and label.
  - The dx and dy attributes represent the trucks' velocities in the x and y directions, respectively.
  - In the simulation loop, the code updates the positions of the trucks based on their current positions and velocities.
  - The code checks the boundaries of the simulation window and changes the velocities accordingly to maintain movement within the defined boundaries.
  - By adjusting the velocities (dx and dy), the code ensures that the trucks move horizontally and vertically while maintaining a certain distance from each other.
- Maintaining Distance: The constant DISTANCE-BETWEEN-TRUCKS represents the desired distance between each pair of consecutive trucks in the simulation. When creating the trucks in the trucks list, the y coordinate for each truck is calculated by adding the DISTANCE-BETWEEN-TRUCKS to the previous truck's y coordinate. This ensures that each truck is vertically positioned below the previous truck by the specified distance. As the simulation progresses, the trucks' positions are updated based on their velocities (dx and dy), ensuring that the desired distance between trucks is maintained. The code checks the boundaries of the simulation window to adjust the trucks' velocities accordingly and prevent them from leaving the defined boundaries. By adjusting the velocities and updating the positions, the simulation maintains the desired distance between the trucks while they move.
  The machine learning part of the code trains a decision tree classifier to predict truck colors based on their characteristics. This prediction is then used to assign the lead truck ID, and the simulation environment ensures that the trucks move while maintaining the specified distance between them.

5.2 Joining Truck (Doluwamu Taiwo Kuye)

In the code, the spacebar is used to accept the request for trucks to join the platoon. Here's how it works:

- The main simulation loop continuously checks for events, including keyboard events.
- When the spacebar (it serves as our request accepted signal given by lead truck) is pressed, the code executes the corresponding logic to handle the joining process.
- The joining process is divided into three steps: truck 3 starts the platoon, truck 2 joins, and finally, truck 1 joins.
- Initially, truck3started, truck2joined, and truck1joined variables are set to False, indicating that no trucks have started or joined yet.
- When the spacebar is pressed for the first time, truck3started becomes True, indicating that truck 3 starts moving.
- When the spacebar is pressed again, the code checks if truck3started is True and truck2joined is False. If so, it sets truck2joined to True, indicating that truck 2 has joined the platoon.
- Similarly, truck 1 joins under the progressive conditions from the last state.
  The spacebar acts as a trigger to accept requests and progress through the joining process step by step. Each press of the spacebar corresponds to a specific action in the platoon formation, allowing trucks to join one after another.

5.3 Obstacle Avoidance (Doluwamo Taiwo Kuye)

The obstacle avoidance simulation shows a simple implementation of truck platooning utilizing an arc-based movement technique. A platoon formation of trucks (represented by different colored rectangles) moves in the simulation. The goal is to avoid an obstacle (represented by a circle) while keeping the platooning formation intact.

Each vehicle has been equipped with sensors that detect the position and size of the barrier. Based on this data, the trucks compute an appropriate arc to avoid the obstruction while maintaining a safe distance. The avoidance maneuver is carried out by progressively shifting the trucks' heads towards a target angle, aligning them with the intended course. The vehicles maintain a steady speed and modify their positions in accordance with the avoidance algorithm. The obstacle stays in place in the center of the screen. (See Appendix C for the simulation results)

6 Conclusions

In this semester, we got this opportunity to work and contribute towards the topic of platooning. From scenario diagrams to simulations we got to learn a great deal from this project. We included several scenarios for our platoon after brainstorming together. Then we implemented the scenarios successfully on UPPAAL, then we chose implemented machine learning algorithm that can decide for the lead truck in our system and then we visualised the simulation along with machine learning algorithm using pygame.

7 Declaration of Originality

I, Arsal Abbasi. I, Doluwamo Taiwo Kuye. I, Mohammad Ashraf Siddiqi herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. I agree that my work may be checked by a plagiarism checker.

8 APPENDICES

8.1 Appendix A

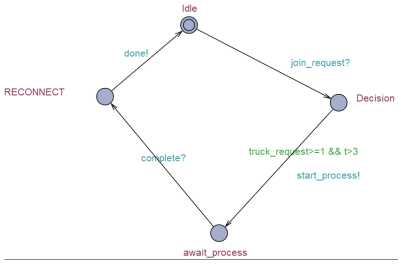


Fig. 9: component automata for truck join

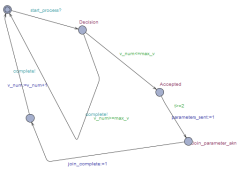


Fig. 10: Truck Join

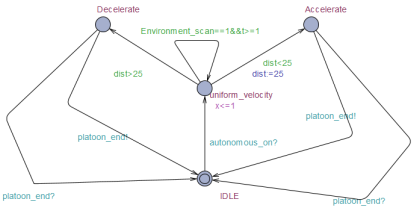


Fig. 11: control lead

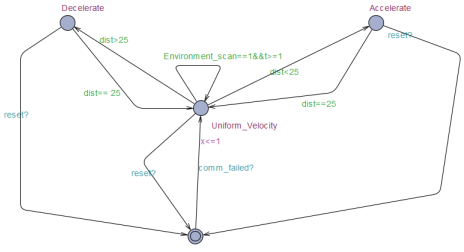


Fig. 12: control follow

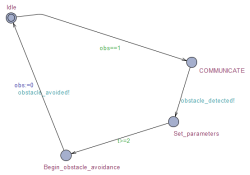


Fig. 13: obstacle avoidance

8.2 Appendix B

```python
# Define colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)

# Draw the simulation environment
screen.fill(WHITE)
pygame.draw.rect(screen, BLACK, (0, 0, SCREEN_WIDTH, BORDER_WIDTH))
pygame.draw.rect(screen, BLACK, (0, SCREEN_HEIGHT - BORDER_WIDTH, SCREEN_WIDTH, BORDER_WIDTH))
pygame.draw.rect(screen, BLACK, (0, BORDER_WIDTH, BORDER_WIDTH, SCREEN_HEIGHT - 2 * BORDER_WIDTH))
pygame.draw.rect(screen, BLACK, (SCREEN_WIDTH - BORDER_WIDTH, BORDER_WIDTH, BORDER_WIDTH, SCREEN_HEIGHT - 2 * BORDER_WIDTH))

# Create trucks
trucks = [
    {
        'id': 0,
        'x': BORDER_WIDTH,                                          # Start from the top-left corner (x-axis)
        'y': BORDER_WIDTH,                                          # Start from the top-left corner (y-axis)
        'dx': TRUCK_SPEED,                                          # Move horizontally (to the right)
        'dy': 0,                                                    # No vertical movement
        'color': TRUCK_COLORS[chosen_truck_id - 3],
        'label': f"ID: 0"
    },
    {
        'id': 1,
        'x': BORDER_WIDTH,                                          # Start from the top-left corner (x-axis)
        'y': BORDER_WIDTH + DISTANCE_BETWEEN_TRUCKS,                 # Start from the top-left corner plus distance (y-ax
        'dx': TRUCK_SPEED,                                          # Move horizontally (to the right)
        'dy': 0,                                                    # No vertical movement
        'color': TRUCK_COLORS[chosen_truck_id - 2],
        'label': f"ID: 1"
    },
    {
```

Fig. 14: Maintaining distance, full code found here.

```python
    }
]

# Create font for labels
font = pygame.font.Font(None, 24)

# Variables for tracking truck joining
truck_3_started = False
truck_2_joined = False
truck_1_joined = False

# Main simulation loop
running = True
clock = pygame.time.Clock()
while running:
    # Limit the frame rate
    clock.tick(60)

    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
            if not truck_3_started:
                truck_3_started = True
            elif truck_3_started and not truck_2_joined:
                truck_2_joined = True
            elif truck_2_joined and not truck_1_joined:
                truck_1_joined = True

    # Update truck positions
    for truck in trucks:
        if truck['id'] == 2 and truck_3_started:
            truck['x'] += truck['dx']
            truck['y'] += truck['dy']
```

Fig. 15: Joining truck, full code found here.

```python
    # Calculate the time elapsed since the start of the simulation
    elapsed_time = pygame.time.get_ticks() - start_time

    # Clear the screen
    screen.fill((255, 255, 255))

    # Update and draw each car in the platoon
    for i in range(len(CAR_COLORS)):
        # Calculate the distance and angle between the car and the obstacle
        distance = math.sqrt((car_x - OBSTACLE_POSITION[0]) ** 2 + (car_y - OBSTACLE_POSITION[1]) ** 2)
        angle = math.degrees(math.atan2(OBSTACLE_POSITION[1] - car_y, OBSTACLE_POSITION[0] - car_x))

        # Check if the car needs to avoid the obstacle
        if distance <= AVOIDANCE_DISTANCE:
            avoidance_angle = angle + AVOIDANCE_ANGLE
            avoidance_x = OBSTACLE_POSITION[0] + AVOIDANCE_DISTANCE * math.cos(math.radians(avoidance_angle))
            avoidance_y = OBSTACLE_POSITION[1] + AVOIDANCE_DISTANCE * math.sin(math.radians(avoidance_angle))
            target_angle = math.degrees(math.atan2(avoidance_y - car_y, avoidance_x - car_x))
        else:
            target_angle = angle

        # Adjust the car's heading towards the target angle
        angle_diff = target_angle - car_headings[i]
        if angle_diff > 180:
            angle_diff -= 360
        elif angle_diff < -180:
            angle_diff += 360
        car_headings[i] += angle_diff * 0.05

        # Move the car based on its heading
        car_x += CAR_SPEED * math.cos(math.radians(car_headings[i]))
        car_y += CAR_SPEED * math.sin(math.radians(car_headings[i]))

        # Draw the car
        car_rect = pygame.Rect(car_x - CAR_SIZE // 2, car_y - CAR_SIZE // 2, CAR_SIZE, CAR_SIZE)
        pygame.draw.rect(screen, CAR_COLORS[i], car_rect)
```

Fig. 16: Obstacle Avoidance, full code found here.

8.3 Appendix C

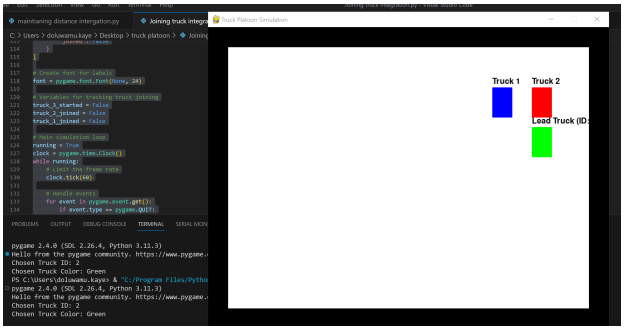


Fig. 17: Joining truck simulation shows the trucks in starting positions for platoon formation

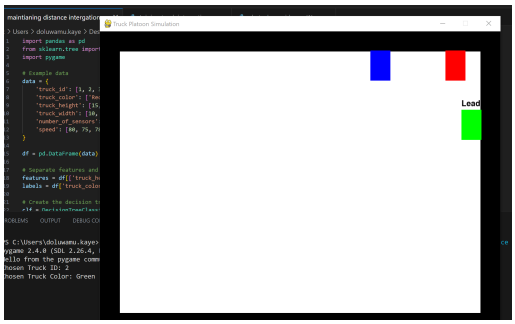


Fig. 18: Maintaining distance simualtion

Fig. 19: Obstacle avoidance simulation