Motivation

On the 1st of June 2022, the 9 Euro ticket was introduced, allowing every person which bought it to travel throughout Germany using regional transport for just 9 Euros per month, from June till August. This has had a clear effect on the way in which people commute and use transportation services, be it for work or school, or just recreationally. Delays, cancellations, accidents and overpopulation are just a few of the noticeable changes. Factors such as time of day, day of the week, state of the passengers and temperature all take their toll in the experience of the people who use public transportation.

The aim of this project was to visualize and simulate the extent to which these factors, together with the dynamic the 9 Euro ticket has introduced, have impacted the way in which people travel.

Report on task 1, Scenario Creation in Vadere

Setup

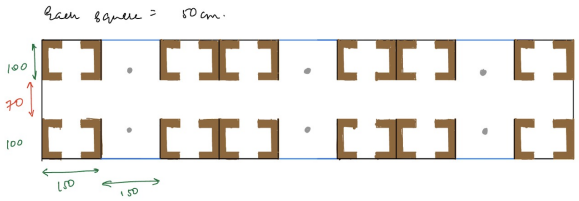
The first task proposed creating two scenarios, representing the topographies of one, respectively two cars of one subway.

Dimensions of the subway cars

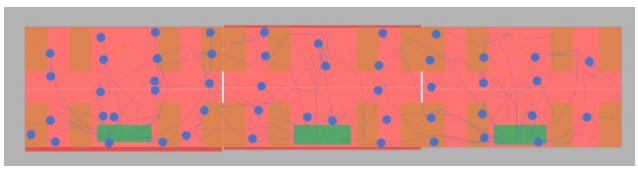
The following measurements represent the dimensions regarding one subway car:

- length: 13.5 m
- width: 2.7 m
- chair size: 50 cm by 50 cm

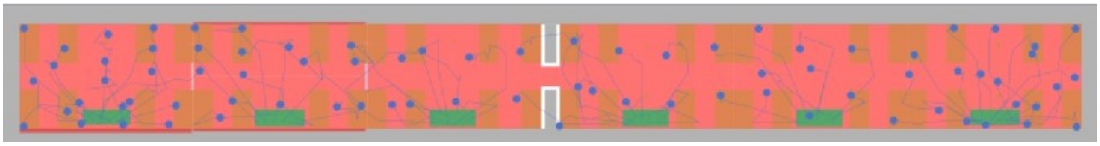
These can also be seen in figure 1a, which is a sketch of a subway car. The actual scenarios can be seen in figures 1b, 1c.



(a) Sketch of one car



(b) One-car scenario



(c) Two-car scenario

Figure 1: Subway car sketch and base scenarios

Apart from the dimensions, there are some other variables known: the total number of seats in a car is 48 (12 units of 4 seats each), and 6 additional bars per car that people can hold on to; each car has 3 doors (left, middle, right). This data was translated into Vadere as follows:

- 54 targets per car, with ID's from 100 to 153 for the first car, and from 200 to 253 for the second car
- 3 doors per car, with ID's from 0 to 2 and 3 to 5 respectively

Algorithm 1 Compute target lists for each source

```
sources_per_car = 3
targets_per_car = 54
targets_per_source = target_per_car/sources_per_car {=18}
targets = [] {empty list}
if source_id ≤ sources_per_car then
  car_id = 200 {second car}
else
  car_id = 100 {first car}
end if
{compute minimum and maximum value for target range}
range_min = car_id + targets_per_source ∗ (source_id%3)
{the last target id for the last source in a car represents a special case}
if last source in car then
  range_max = car_id + targets_per_car
else
  range_max = car_id + targets_per_source ∗ ((source_id + 1)%3)
end if
for i in range(range_min, range_max) do
  add target ID to list
end for
return targets
```

Between the two cars, two obstacles were added, creating a bottleneck. Each source has a list of 18 targets, as people coming out of a certain gate tend to go to the seats closest to that gate. The aforementioned ID's and lists were set using the file *modify_ids.py*; the procedure was as follows (algorithm 1).

Applying this algorithm, we get the following target lists (table 1):

Table 1: Target IDs for each source

Source ID	Car	Target list
0	First	[100,...,117]
1		[118,...,135]
2		[136,...,153]
3	Second	[200,...,217]
4		[218,...,235]
5		[236,...,253]

Now, two base scenarios were created, which will later be modified for testing.

The red areas which can be seen in the figures are measurement areas. Each car has 6 areas (so 12 areas in total for a two-car scenario) which provide the data for density and passengers trajectories, later used in the machine learning part of the project.

Another value which proved to be important to the development of the project was the personal space of each pedestrian. This had to be modified accordingly so that it bests simulates the way in which the passengers move. If too high, they will stop after a few timesteps if the subway has too many people in it. The value should be smaller when it's more crowded.

Report on task 2, Temperature influence

The problem that is raised in this task is the probability of failure of the air conditioning in one of the two cars, which forces the passengers to look for a better ventilated spot throughout the subway.

Setup

The way in which the change in temperature was simulated, using the tools already present in Vadere, is as follows: assign multiple targets to a source and modify the waiting time of all targets in the first car. In this way, people will move from the first car (left) to the second one (right).

Each target can be treated as a traffic light, with the waiting time representing for how long is the target active/free. The shorter the waiting time is, the higher the temperature in the car, forcing the passengers to look for available targets in the other car. In our experiments, we simulate two different temperature values by the time air conditioner stops working. Outside temperature of 30 degrees corresponds to the waiting time of 5.0 seconds and a temperature of 26 degrees corresponds to 10.0 seconds.

This was done in the *generate_scenario.py* with the following code:

```
variables = {
    "waitingTime": 5.0,
    "spawnNumber": spawnNumber,
    "minimumSpeed": 0.01,
    "maximumSpeed": 1.3
}
targets_per_car = 54
for i in range(targets_per_car):
    data["scenario"]["topography"]["targets"][i]["waitingTime"] = variables["waitingTime"]
```

This would translate as follows: for each of the targets in the first car, assign a waiting time of 5 seconds (or 10 seconds). A more detailed presentation of the scenario generation file can be found in task 5.

Simulation

As mentioned before, to simulate a fault in the air conditioning system, the waiting time has to be changed. First, the waiting time was set to 5.0 seconds, which made the passengers walk towards the other car after 5 seconds, leaving a few to stay in the original car, as seen in figure 2. The corresponding video can be found in *demos/Task_2_time_5*.

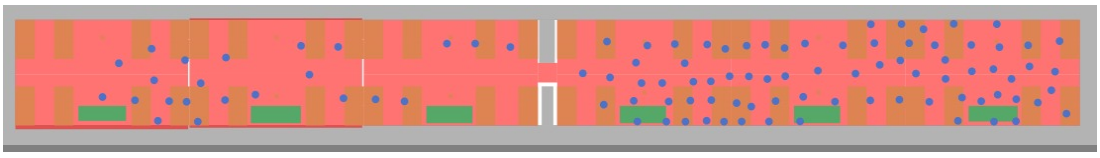


Figure 2: Two-car scenario with waiting time = 5 seconds

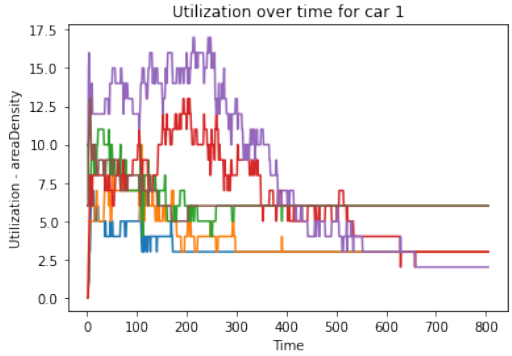
However, when setting the waiting time to 10.0 seconds, we observe that the passengers move faster towards the other car, which can be observed in *demos/Task_2_time_10*.

Therefore, we can conclude from this task that making all the pedestrians move once they get in to the next car might make it more difficult for them to get to the cooler car, since they will form a crowd at the bottleneck entrance. The data generated in this task was used in the following machine learning task.

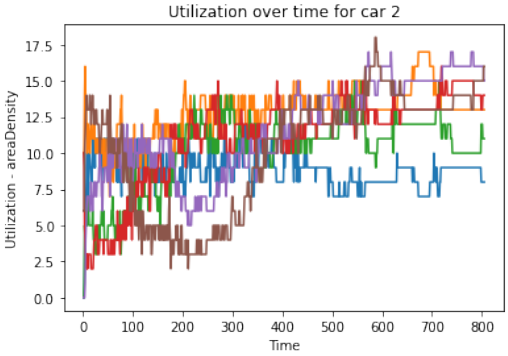
A detailed look into the utilization

The default time is set to 5.0 seconds, and we plotted the utilization of the subway cars in the graphs seen in figures 3, 4, generated in the Jupyter notebook "*graphs_task_2*". The overall utilization of the two cars can be seen in figure 3. It is clear to see that the utilization of the first car is dropping steadily, as the passengers move continuously towards the second car, making its utilization constantly increase.

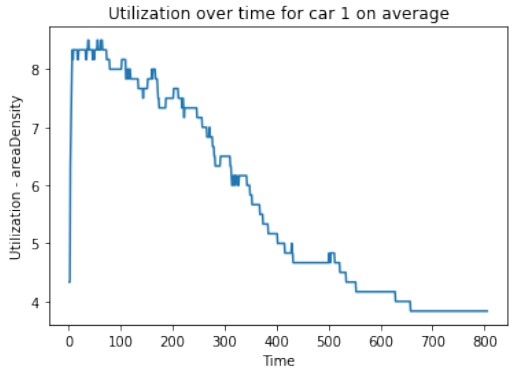
When setting the time to 10 seconds, it can be observed that it takes more time for the passengers to get to the second car, as the graph lines in figures 3e, 3f show: there is a slower decrease in the first car, and increase in the second car, in the area density. The timesteps on the Ox axis suggest the same. When the waiting time is 5 seconds, the movement of the passengers has already stopped by timestep 700, whereas for a waiting time of 10 seconds, at timestep 1200 the passengers still move.



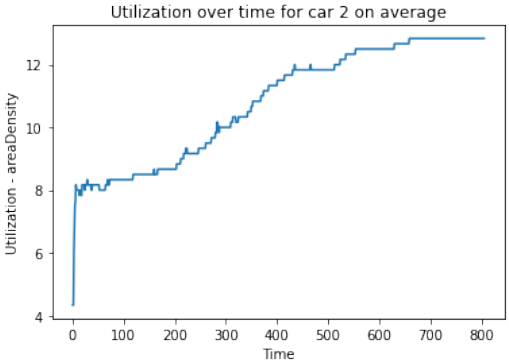
(a) First car all areas



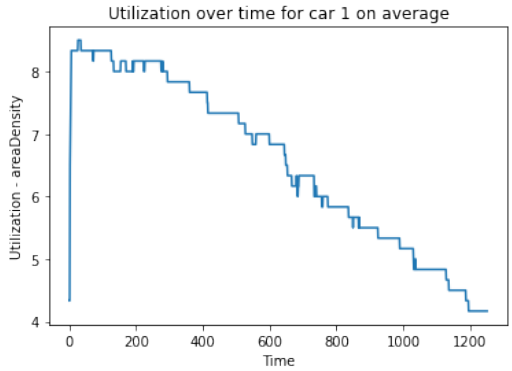
(b) Second car all areas



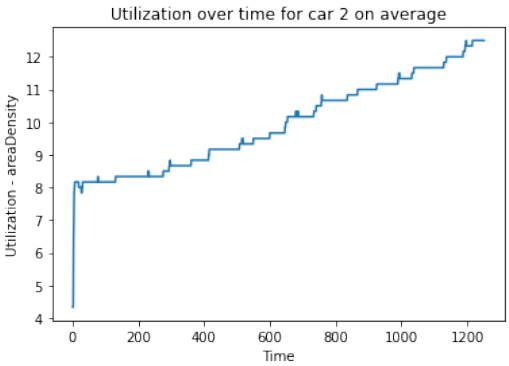
(c) First car average



(d) Second car average



(e) First car average with 10 seconds



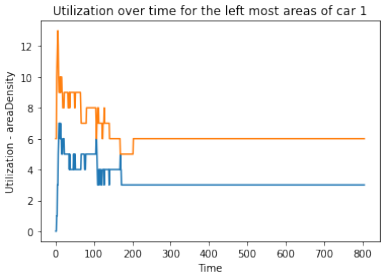
(f) Second car average with 10 seconds

Figure 3: Complete and average utilizations

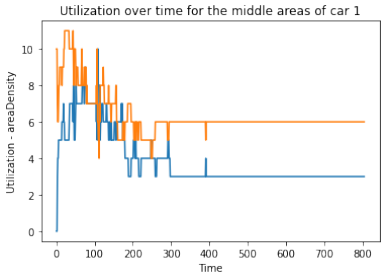
Then, we divided each car into 3 parts, left, middle and right, each containing two measurement areas, hence the two differently coloured lines in the graphs, which show a mirror-like development. The utilization of these areas can be seen in figure 4.

Looking at figure 4a, the utilization of the left area of the first car can be observed: it plummeted, and then remained constant over a long period of time (from timestep 200 all the way to the end). The middle part of the first car shows more traffic for the first half of the simulation (later than the left part, timestep 300), and then also steadiness for the rest of the time in the area density. The rightmost part of the first car shows the most traffic, as that's where the bottleneck-pass towards the second car is. Therefore, it is explained why the utilization gets stable later than the first two parts of the car (only by timestep 700).

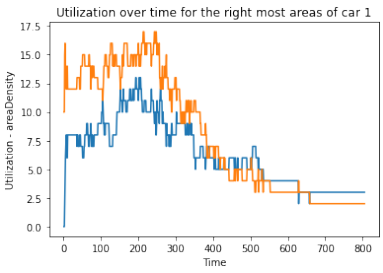
The second row of the figure table shows the second car. The leftmost areas show a constant growth in the area density, as people continuously pass from the bottleneck. A steady trend is shown only starting from timestep 750. The middle areas also show a boost in the area densities. Between timesteps 0 and 300, there is a more abrupt escalation, afterwards the tendency is also more or less stable. The rightmost part of the second car shows an initial decrease, and then an upsurge, as more people come in the second car, and force the other passengers (which were already in the second car) to move and accommodate the ones who come.



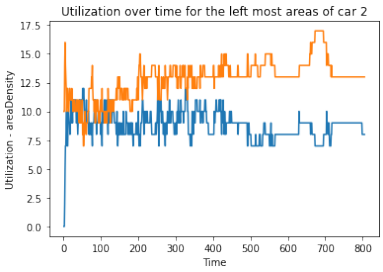
(a) First car left areas



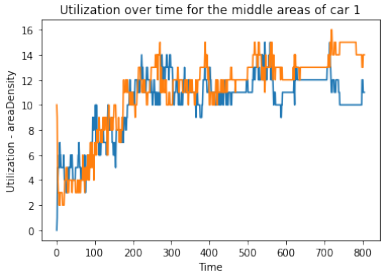
(b) First car middle areas



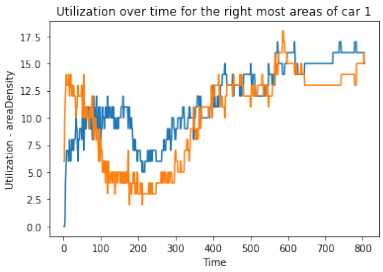
(c) First car right areas



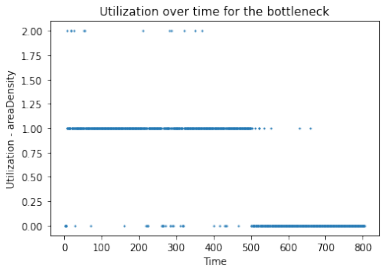
(d) Second car left areas



(e) Second car middle areas



(f) Second car right areas



(g) Bottleneck

Figure 4: Utilizations

Report on task 3, Dynamic Obstacle Avoidance

In task 3 we implemented a basic version of dynamic obstacle avoidance of a certain group of moving passengers. Although achieved behaviour does not represent perfect avoidance of moving obstacles and does not cover all the edge cases, we think that this is a valuable asset to the existing Vadere software. Dynamic obstacle avoidance is modelled with the custom SIRO model for Vadere.

Setup

We create our custom submodel for Vadere, similar to the SIR from assignment 2. The SIRO group model was used to simulate the different groups of passengers:

- S: **S**ober - blue
- I: **I**ntoxicated - red
- R: **R**egurgitating/**R**elieved - black

An additional 'O' stands for obstacles as we treat a passenger that has regurgitated as a moving object that non-regurgitating passengers try to avoid. Sober people are not aggressive and **cannot** become intoxicated (in contrast to the original SIR model).

The number of intoxicated people at start is defined by the "intoxicatedAtStart" attribute and can only decrease during the simulation. Intoxicated people can become regurgitating/relieved with a certain probability, which is defined by the "reliefRate" attribute. You can also change the number of relieved people in the beginning by adapting the "relievedAtStart" attribute. The last attribute is "obstacleRadius", which defines what distance everyone should keep to the regurgitating pedestrians. The default settings are listed below:

```
"org.vadere.state.attributes.models.AttributesSIROG" : {
       "intoxicatedAtStart" : 10,
       "relievedAtStart" : 0,
       "reliefRate" : 0.003,
       "obstacleRadius" : 3.0
     }
```

In order to create obstacles "on the go", we implemented **dynamical obstacle creation**. Once a person regurgitates, their neighbours are checked to see if they are too close to that person within a certain radius. If this is the case, we shift the position of that neighbour in the opposite direction from the R person.

Shifting is done in the update step of the model as follows: given two sets of coordinates of the two passengers, the vector which represents the difference is computed and then normalized. Since this is a vector, it already has a direction, so we just add it to the initial coordinates of the neighbour, hence making them "run away" from the person who vomited. Since we inspect the obstacleRadius in every update step, this enables to evade obstacles even if they change their initial position (dynamic obstacles).

Changing the group from intoxicated to relieved is done in the same way as we implemented it in assignment 2 for the original SIR model.

This result of the simulation can be seen in figure 5. For this, the following values were implemented in the scenario titled "*one_car_task_3*".

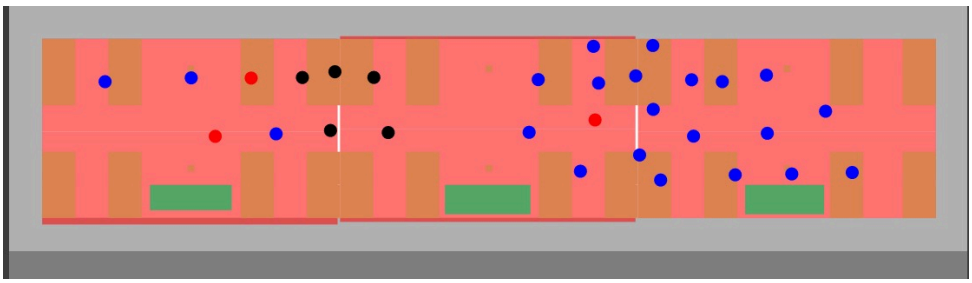


Figure 5: One-car scenario with obstacle avoidance

As we can see from the picture, sober passengers are avoiding the relieved ones as expected. The video of the simulation can be found in the *demos* folder under the name $Task\_3$. The SIRO group model implementation and the corresponding attributes can be found in the folder $task\_3$. The model should be added to the Vadere source code and executed as usual.

Report on task 4, Machine Learning

Setup

The current task aims to predict the utilization of the subway cars, by anticipating how much will each of the areas densities be, and the passengers' trajectories.

Output data

Every time a scenario is run, a new output folder is created, containing data about the simulation. In order to collect the information needed to train the model, new processors were added. In figure 6 one can see which files are returned after the successful run of a simulation.

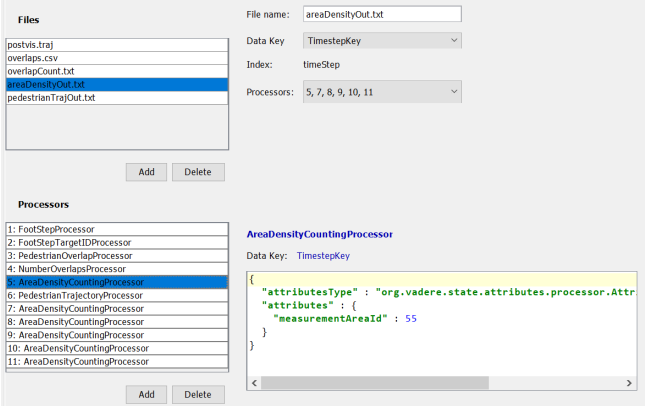


Figure 6: Vadere output files

Density processors

A processor representing each measurement area was added in order to gather each density at every timestep, so there are 6 and 12 density processors respectively. The processors all have as key the timestep. Figure 7 shows one output file containing the densities throughout the simulation. The structure of the file can be better observed in table 2.

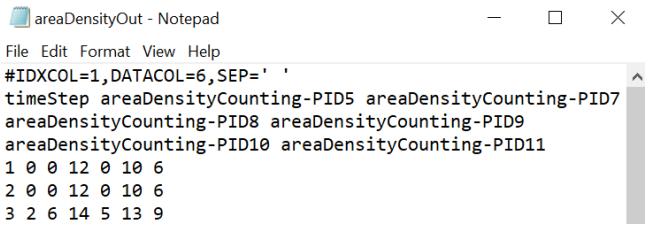


Figure 7: Area densities output file

Table 2: Structure of densities output file

Timestep	Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
1	0	0	12	0	10	6
2	0	0	12	0	10	6
3	2	6	14	5	13	9
...	...	...	...	...	...	...

Trajectory processor

In order to get the trajectory of the passengers, one processor takes as key the passenger ID and then takes their start and finish positions at every timestep. The file can be seen in figure 8, and the structure of the file is as follows:

- list containing passenger IDs
- for each passenger ID:
  - "footSteps" dictionary containing:
    - start time of the timestep
    - stop time of the timestep
    - dictionary with x and y coordinates at start of timestep
    - dictionary with x and y coordinates at stop of timestep

Figure 8: Excerpt from trajectory output file

Predicting the utilization

For this task we try to estimate the utilization of certain regions of interest (ROI) within subway cars. For our tests we have considered the following regions:

1. Front of the car
2. Middle/Center of the car
3. Right of the car
4. Passage between 2 cars (i.e This is the bottleneck of our model)

In the start of each simulation, we assume a fixed number n of passengers to be intoxicated before entering the car. We then run the simulation to obtain the per area density for 6 ROIs in each subway car. As the train starts to move, we assume that the intoxicated passengers can be sick with a probability ρ, causing the other non-intoxicated passengers to actively avoid them. This situation will lead to crowding in some parts of the car, which will ultimately result in a situation where the passengers will be angry.

The goal of this task is to predict the behaviour of the passengers after they have entered into the subway car. We define a maximum threshold for the area density, which if exceeded will cause the passengers to become angry.

In our approach we perform a time-series prediction using the Radial Basis function to predict the average car utilization over time.

We pick the threshold for the maximum allowable area density as ¿12. If the average areaDensity in any region of the car is above this value we consider this to be a situation where the passengers will get angry due to overcrowding.

Research Question

Can the given number of people be safely accommodated in a single train car, given that n of them are intoxicated?

What is the critical number of people who can be allowed to travel without having a conflict?

Predicting the number of people that can be accommodated on the transport

Algorithm 2 Algorithm to approximate the future utilization

Create embedding space of window size M.
Apply PCA to the embedding space. Since each car has 6 ROIs where we are monitoring the areaDensity, we choose no of PCA components =3.
Non-linear approximation of Velocity using radial basis function.
Approximate the utilization using radial basis function.
return predictions

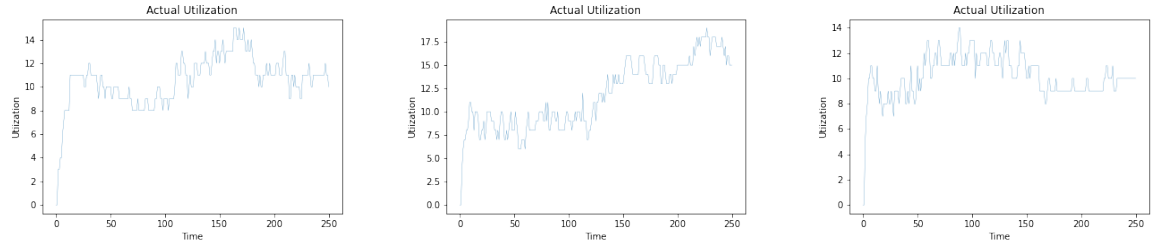
Predictions of utilization in the SIRO scenario

Now, the model was used to predict the utilization of one subway car when there are 5, 8 and 10 intoxicated people. The actual utilization graphs can be seen in figures 9, 13 and 11 respectively, and the future predicted utilization in graphs 10, 14, 12.

The actual predictions show the density of the pedestrians from the time they enter the car until they settle down. In the initial time steps the utilization appears to be lower and then settles at a stable value as the car becomes more and more full.

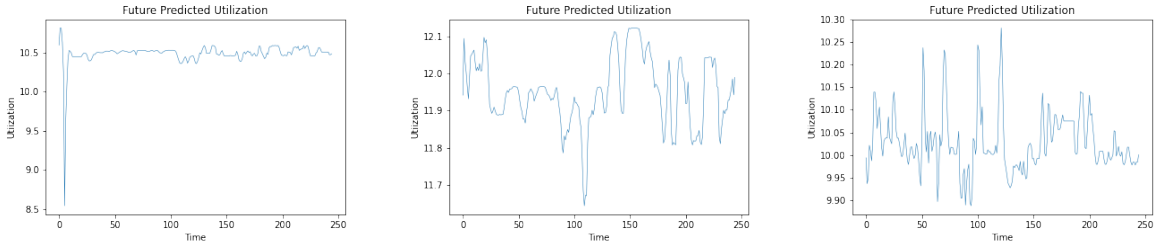
The predictions show the future behaviour of the passengers, hence there might be less of a resemblance between the true observed movement of passengers and the outcome of the model.

Looking at the graphs with 5 intoxicated people, we can see that the predictions are more or less similar to the true utilizations. The left areas don't show much variance, like the middle area does.



(a) Left areas (b) Middle areas (c) Right areas

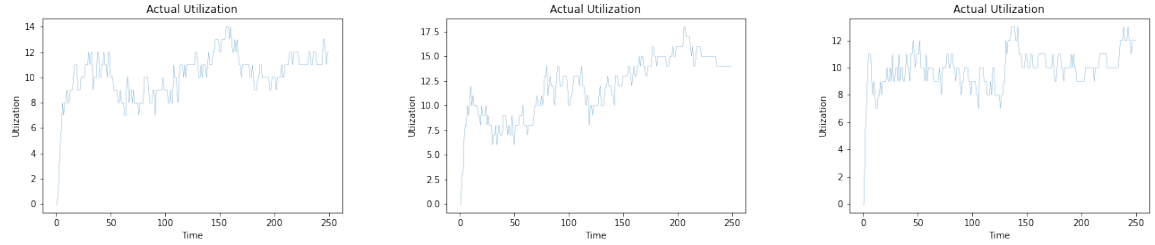
Figure 9: True utilization of one car with 5 intoxicated people



(a) Left areas (b) Middle areas (c) Right areas

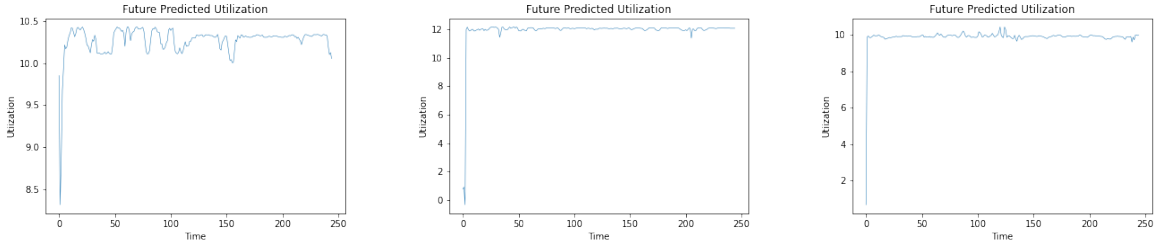
Figure 10: Predicted utilizations of one car with 5 intoxicated people

The 7 intoxicated people scenario shows more variance in the utilization of the left area. The middle area presents a steady growth, while the right area is unstable.



(a) Left areas (b) Middle areas (c) Right areas

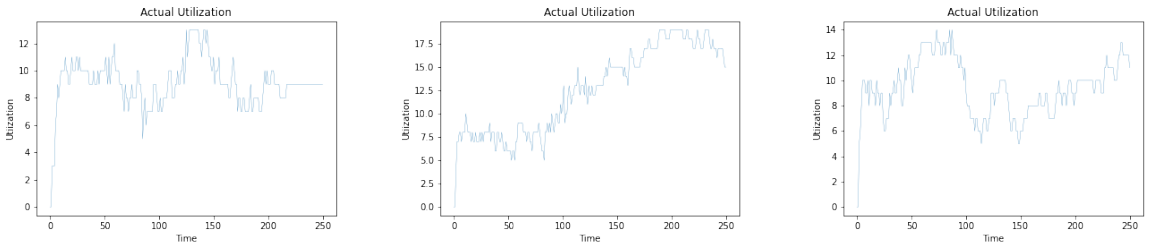
Figure 11: True utilizations of one car with 7 intoxicated people



(a) Left areas (b) Middle areas (c) Right areas

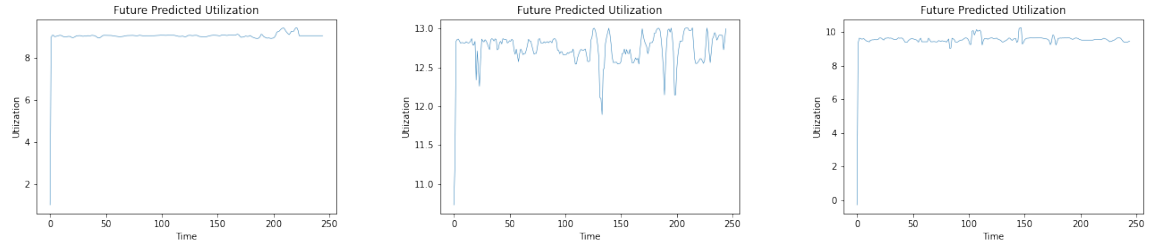
Figure 12: Predicted utilizations of one car with 7 intoxicated people

Having 8 intoxicated people increased the area density in the middle of the car. The left area becomes stable after timestep 200, while the right area shows an initial boost, then a decrease and a slow increase.



(a) Left areas (b) Middle areas (c) Right areas

Figure 13: True utilizations of one car with 8 intoxicated people



(a) Left areas (b) Middle areas (c) Right areas

Figure 14: Predicted utilizations of one car with 8 intoxicated people

From the above graphs we can see that when the number of intoxicated people in the train car reaches 8, there is overcrowding and the areaDensity near the middle exceeds 12. This would cause a conflict. In our simulation scenario 14b, we cannot fit more than 8 intoxicated people in a single subway without the risk of a conflict happening.

Report on task 5, Testing

Setup

The last task aimed to show how the simulation would look like in different scenarios, with discrete and contrasting parameters. In order to make it easier to create such scenarios, the Python script *generate_scenario.py* was created.

Generating a new scenario

In order to generate a new scenario, these are the steps which need to be taken:

- choose number of cars (1 or 2)
  - base scenario will be selected
- choose how many passengers should enter the subway
  - a list with the distribution of the passengers over the gates will be generated
- modify target waiting times
- update passengers' speeds
- create a new scenario file

Choose number of cars

As written before, the number of cars in a scenario can be either one or two. Based on this, the base scenario will either be "one_car.scenario" or "two_cars.scenario". This is done using the following code:

```
number_of_cars = 1

file1 = "./output_files_vadere/final_project/scenarios/one_car.scenario"
file2 = "./output_files_vadere/final_project/scenarios/two_cars.scenario"

file = file1 if number_of_cars == 1 else file2
```

Choose number of passengers

At the beginning, a desired total number of passengers is given. Then, algorithm 3 shows how they are divided among the number of sources. On a regular basis, there are more passengers at the ends of the subway (first and last gates, their IDs stored in *busierGates* list), since the entrance to the subway station are at both ends, usually.

Therefore, we use a Normal distribution to sample how many passengers come out through each gate with standard deviation 1 and a mean defined as follows:

- if the gate is first or last, then we expect to have a number between half and the mean of the total number expected over the gates

$$\frac{1}{2}(\frac{1}{2} * \text{total number} + \frac{1}{\text{gates}} * \text{total number}) = \text{total number} * \frac{2 + \text{gates}}{4 * \text{gates}} \tag{1}$$

- else, we expect between 0 and the mean of the total number expected over the gates

$$\frac{1}{2}(0 + \frac{1}{\text{gates}} * \text{total number}) = \text{total number} * \frac{1}{2 * \text{gates}} \tag{2}$$

In code, the mean is created as follows:

```
meanFactor = (2 + gates) / (4 * gates) if gate in busierGates else 1 / (2 * gates)

mu = totalNumber * meanFactor
```

Algorithm 3 Divide passengers among the gates

```
gates_per_car = 3
set total number of passengers
gates = gates_per_car * number_of_cars {= 3 or 6}
spawnNumber = [0] * gates
busierGates = [0, gates − 1]
checkPassSum = 0
lastBusyGate=-1
for all gate in gates do
  create normal distribution with standard deviation 1 and mean computed as seen in equations 1,2
  x = random number sampled from distribution
  if current gate is the first then
    lastBusyGate = x
  else if x < 0 or x >the population entering at the first gate then
    resample x
  end if
  simulate what would happen if we added the current x to the gate
  if total number of passengers is achieved then
    spawnNumber[gate]=x
    break
  else if current gate is not first then
    check how many passengers are left to be added
    difference = total number - current sum
    if (current gate is last and we need more people) OR the number of people is exceeded then
      x = x + difference {compute x to satisfy these requirements}
    end if
  end if
  checkPassSum += x {add final computed x to the sum}
  spawnNumber[gate] = x
end for
return spawnNumber
```

The dictionary called "variables" is added in order to contain all the necessary variables which are to be modified:

```
variables = {
    "waitingTime": 2.0,
    "spawnNumber": spawnNumber,
    "minimumSpeed": 0.01,
    "maximumSpeed": 1.3
}
```

Modify target waiting times

This is used mainly in task 2, where the goal is to simulate malfunctioning air conditioning. The code below shows how the waiting times are modified for each of the targets in the first of the two cars:

```
targets_per_car = 54
for i in range(targets_per_car):
    data["scenario"]["topography"]["targets"][i]["waitingTime"] = variables["waitingTime"]
```

Update passengers' speeds

The minimum and maximum speeds of the passengers can also be modified using the following code:

```
data["scenario"]["topography"]["attributesPedestrian"]["minimumSpeed"] = variables["
    ↪ minimumSpeed"]
data["scenario"]["topography"]["attributesPedestrian"]["maximumSpeed"] = variables["
    ↪ maximumSpeed"]
```

Update SIRO attributes

This is specific to task 3. We added a new dictionary called "variablesSIRO" such that they are not mixed up with the others. With this, the attributes of the SIRO model can be modified:

```
variablesSIRO = {
    "intoxicatedAtStart": 7,
    "relievedAtStart": 0,
    "reliefRate": 0.003,
    "obstacleRadius": 3.0
}
for key in variablesSIRO.keys():
    data["scenario"]["attributesModel"]["org.vadere.state.attributes.models.AttributesSIROG
        ↪ "][key] = variablesSIRO[key]
```

Create a new scenario file

After all the variables were updated, a new scenario file can finally be generated. In order for it to be clear, the following naming convention was implemented:

$$\text{file path} + \text{number of cars} + \text{time and date} + \text{".scenario" extension} \tag{3}$$

The file path locates where the scenario will be saved, which is in the output files folder, together with the other scenarios ("$output\_files\_vadere/final\_project/scenarios/$").

The part which makes the name unique is the middle section. Based on how many cars there are, either the substring "$one\_car\_$" or "$two\_cars\_$" is added, followed by the date and time of the creation of the scenario file.

Tests and conclusion

Testing occurred in every task, as we came up with different scenarios for each of those. We also added videos to show the movement and behaviour of the passengers.

We can conclude that it is important to be able to model and predict passenger behavior in order to ensure safe transport, without conflicts and working machinery. We observed the influence factors such like temperature, crowd population and status of passengers can have over the space occupation in the subway.