TMM Lab 04: Microscopic Modelling (17 October 2025)

a.johansson@bristol.ac.uk

Introduction

This lab sheet assumes familiarity with the Microscopic modelling material shared on Blackboard.

There is a Python code provided on Blackboard `tmm_lab_04_code_shared.zip` that you are encouraged to use, and expand on, as a starting point for your own answers.

We would recommend that you use the provided Python code. However, if you have a strong preference for using MATLAB, then you are free to use MATLAB instead.

How to create new modelling scenarios

The Python code mentioned above contains reusable pedestrian modelling code that can be used to define different modelling scenarios. A pedestrian modelling scenario is defined by the following elements:

- `space`: A Python dictionary containing components that make up the walkable space. The following types are supported:
  - `"type": "rectangle", "coordinates": [x0, y0, x1, y1]`
  - `"type": "polygon", "coordinates": [[node1_x, node1_y], [node2_x, node2_y], ..., [nodeN_x, nodeN_y]]`
- `pedestrians`: Pedestrians are defined as groups of pedestrians, each with different attributes. You need to define a unique name for each group, and then define the following attributes:
  - `source`: A named spatial element (rectangle or polygon), used as a source to create new pedestrians of this type.
  - `destination`: A named spatial element (rectangle or polygon), used as a destination for pedestrians of this type.
  - `colour`: A named colour for this pedestrian type.
  - `birth_rate`: Numbers of pedestrians generated per second
  - `max_count`: The maximum number of pedestrians to be generated of this type.
- `boundaries`: A list of line segments that act as boundaries / walls, defined as: `[x0, y0, x1, y1]`.
- `periodic_boundaries`: Optional functionality to add periodic boundaries to your model. The following attributes need to be defined:
  - `axis`: "x" or "y" (depending on whether the period boundary is horizontal or vertical).
  - `pos1`: Position of the left (or top) boundary.
  - `pos2`: Position of the right (or bottom) boundary.
- `functions`: This lets you define your own functionality to plug into the model. The following functions are supported:
  - `update_directions`: This is called for each time step, in order to allow you to update pedestrians' desired direction.

- `pedestrian_initialisation`: This is called each time a new pedestrian is created, in order to allow you to dynamically adjust individual pedestrian properties upon creation of new pedestrians.
- `process_interactions`: This is called on each time step and defines the pairwise forces (pedestrian vs pedestrian, and pedestrian vs boundary).

Hints:

- Press ESC to stop the model at any point.
- If you run the code in a Notebook, the graphical output will not work. Therefore, run it from a terminal instead: `python scenario_corridor.py`
- If you run the `world.render()` functon for each model update, the simulation will run very slowly. Therefore, it is recommended to run the `world.render()` function once for every 10th or 20th model update step.
- You can obtain useful statistics from a simulation by calling `world.get_statistics()`. The statistics is collected in a Python dictionary containing the following parts:
  - `source`: A list of simulation times (in seconds) of when new pedestrians were introduced in the simulation.
  - `transition_time`: A list of transition times (in seconds) for pedestrians to move from one polygon/rectangle to another one.
  - `completed_journeys`: A list of journey times (in seconds) for pedestrians to walk from their source to their destination.

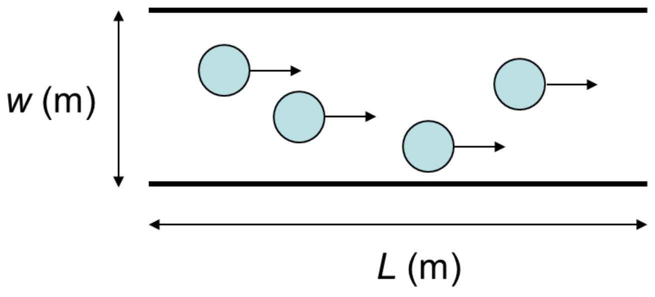
Example: A uni-directional corridor with periodic boundary conditions (cf. scenario_corridor.py):

```
# Define modelling scenario
w = 3
L = 10
world_definition = {
    "space": {
        "corridor": {"type": "rectangle", "coordinates": [0, 0, L, w],
            "colour": "gray", "add_boundaries": False},
        "right": {"type": "rectangle", "coordinates": [L+1, 0, L+2, w],
            "colour": "green", "add_boundaries": False},
    },
    "pedestrians": {
        "group1": {"source": "corridor", "destination": "right",
            "colour": "red", "birth_rate": 99, "max_count": 10}
    },
    "boundaries": [[0, 0, L, 0], [0, w, L, w]],
    "periodic_boundaries": {"axis": "x", "pos1": 0, "pos2": L},
    "functions": {
        "update_directions": update_directions,
        "process_interactions": process_interactions,
        "pedestrian_initialisation": pedestrian_initialisation
    }
}
world = World(world_definition)
```

```
    for i in range(1800):
        world.update(0.05)      # Updades the model, using dt=0.05
        if i%20==0:
            world.render()      # Renders the scene (i.e. visual output)
```

Problem 1: Corridor model

Based on the provided example `scenario_corridor.py`, simulate the following scenarios.



`(a)`. Use the following model parameters:

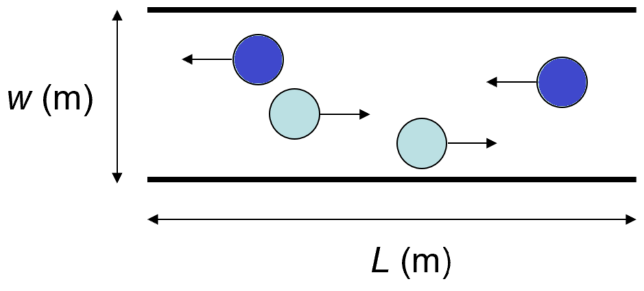
- L = 10 m
- w = 3 m
- N = Number of pedestrians

`(a) (i)`. Run the simulation for a few different values of N, and measure the flow rate. Do the density and flow values broadly look like expected (cf. lecture slides)? If not, can you adjust the modelling parameters so that the results are more in line with expected values?

`(b)`. Modify the corridor simulation according to the following specification:

- L = 50 m
- w = 4 m
- N = 20
- 4 of the 20 pedestrians belong to the same group, and will therefore need an attractive force to keep them together as a group.

`(c)`. Modify the scenario above to create a bi-directional corridor.



`(c) (i)`. With the following modelling parameters:

- L = 10 m

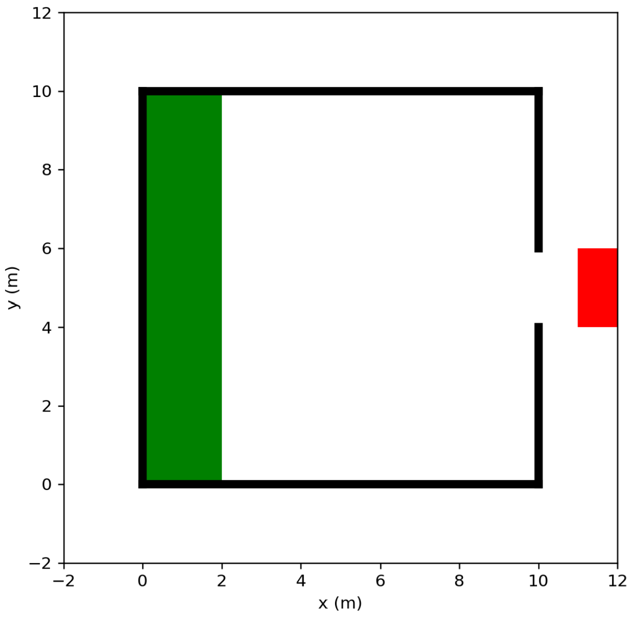
- w = 2 m
- N = 40 pedestrians
- α = Fraction of pedestrians walking from left-to-right (e.g. for α=0, everyone walks from right to left, and for α=1, everyone walks from left to right).

Investigate how the average walking speed will look like as a function of α.

Problem 2: Evacuation modelling

`(a)`. Create a model of the following scenario:

- A room with dimensions 10m x 10m.
- The green polygon shows the source for the pedestrians.
- The red polygon shows the destination of the pedestrians.
- An exit door at the right-hand side.



`(a) (i)`. How long time does it take to evacuate 50 pedestrians through a 2-metre wide door?

`(a) (ii)`. Repeat the experiment for different widths of the exit doorway and find the relationship between door width and evacuation time.

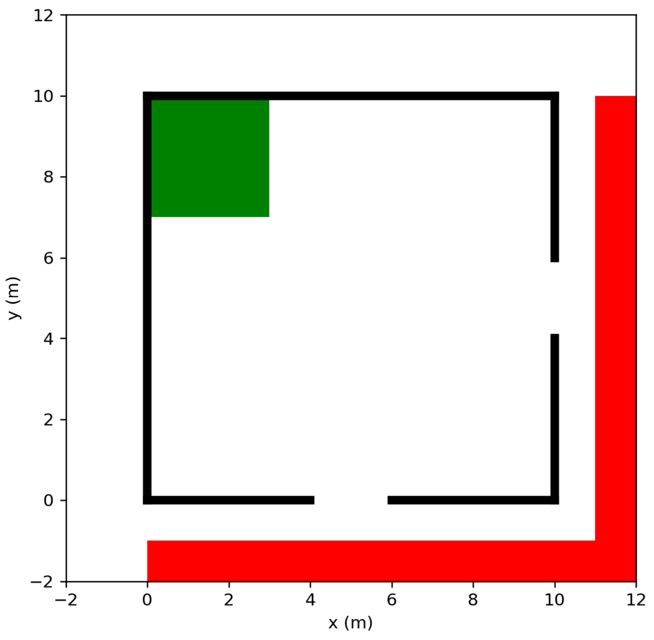
`(a) (iii)`. Run the simulation for three different door widths of 2m, 1.5m, and 1.0m and collect data on the 'time headways', i.e. the evacuation time of pedestian i minus the evacuation time of pedestrian i-1.

Measure the mean, μ, and standard deviation, σ of the time headways for each of the three cases (i.e. for door widths 2m, 1.5m and 1.0m).

Explain the difference in the coefficient of variation (i.e. σ / μ) between the three cases (e.g. by visual observation of the evacuation simulations).

Hint: You can obtain the relevant statistics from completed journeys by calling `world.get_statistics()` or by calling `world.get_statistics(["completed_journeys", "start", "exit"])` (replace `start` and `exit` with the names you have given the green and the red polygons).

`(b)`. Create a model of the following scenario:

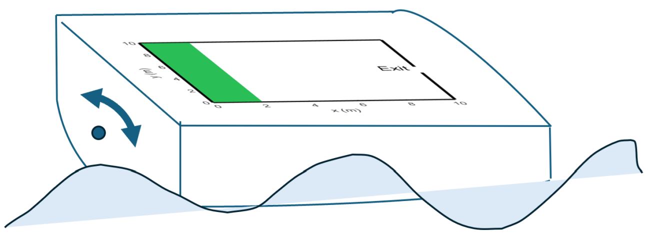


- A room with dimensions 10m x 10m.
- The green polygon shows the source for the pedestrians.
- The red polygon shows the destination of the pedestrians.
- There are two exit doors; one at the right-hand side and one at the bottom.
- Pedestrians should use their closest exit.

`(b) (i)`. How does the evacuation time compare to the single-door scenario above? Compare evacuation times for the one-door vs the two-door scenarios for different exit door widths.

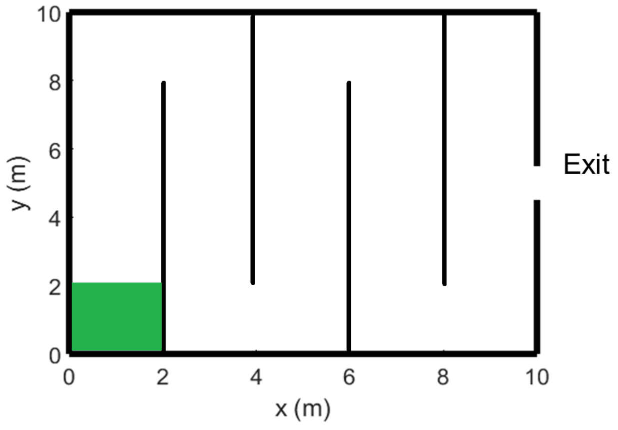
`(c)`. Create a model of the following scenario:

Imagine that scenario `(a)` occurs on a rolling ship (as opposed to on steady ground), with a cycle time of 10 seconds. The effect will be a periodic tilt of the floor in the y direction.

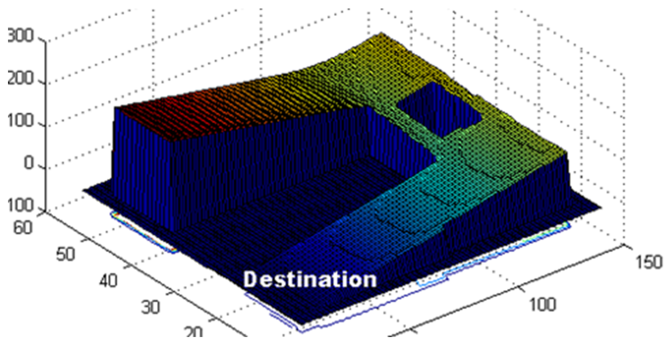


`(c) (i)`. Investigate and describe the impact of the rolling motion on the evacuation.

`(d)` [CHALLENGING]. Create a model of the following scenario:



Note that you can either extend the method used in the previous scenarios, or you can implement a different way-finding method based on the 'potential field' method discussed in the lecture:

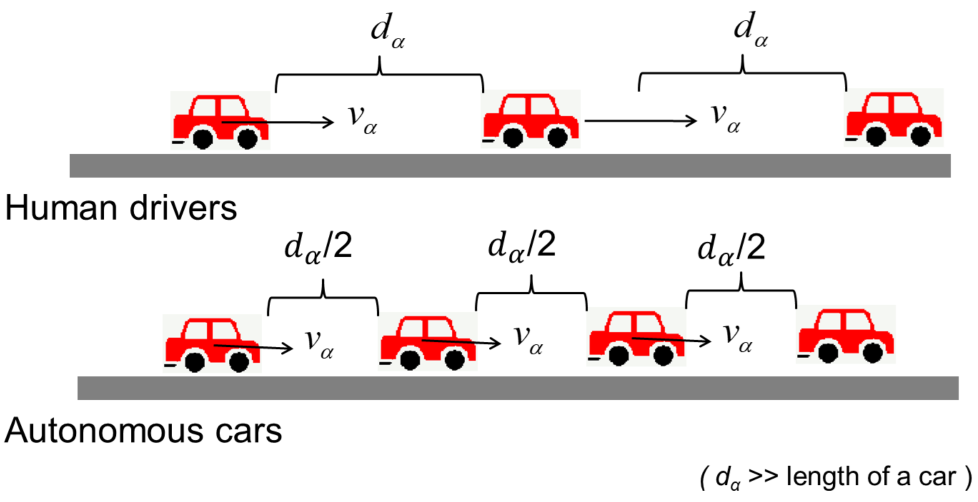


Problem 3: Autonomous cars

Assume that autonomous cars have a quicker reaction time than human drivers, which means that they can therefore halve their distance to the car in front of them, while maintaining the same speed.

For the sake of simplicity, let us assume that the distance between cars d_α >> L, where L is the length of a car.

Therefore, half the distance between cars means twice the density of cars, which means twice the road capacity (cars per second).



`(a)`. Investigate the impact on road capacity when regular cars are gradually being replaced by autonomous cars.

As established above, when 100% of the cars are autonomous, the road capacity is doubled, i.e. it is 100% higher than for human drivers. For a mixed fleet (p autonomous and $1 - p$ regular cars), what fraction p is needed to reach an overall 25% road capacity increase, for the following scenarios:

- `(a) (i)`. *Extreme* case: Each autonomous car keeps a distance of $d_\alpha/2$ to the car in front.
- `(a) (ii)`. *Safe* case: Autonomous cars keep a distance $d_\alpha/2$ to the car in front but only if the car immediately behind it is also autonomous (the distance is d_α otherwise).
- `(a) (iii)`. *Super safe* case: Autonomous cars keep a distance $d_\alpha/2$ to the car in front but only if both the car immediately behind, *and* in front are also autonomous (the distance is d_α otherwise).

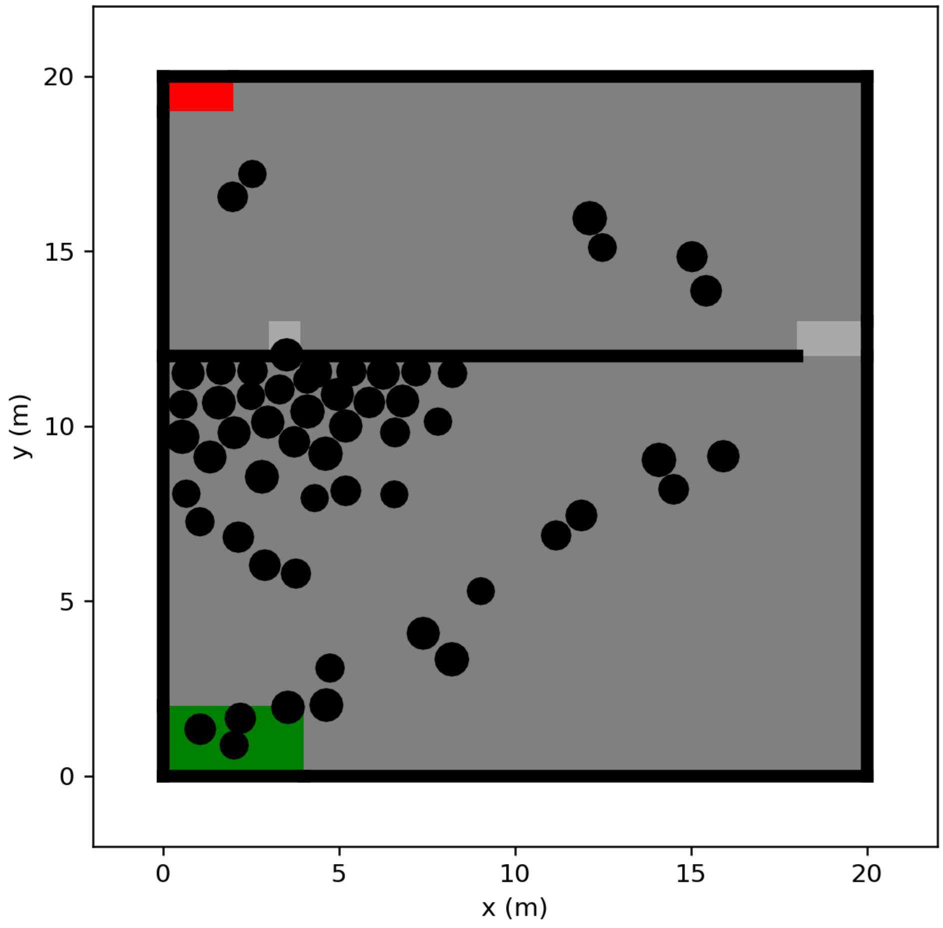
`(b)`. Modify the *Optimal Velocity Model (OVM)* `simulate_cars_ovm.py` to model the scenarios above. Run the model for the following scenario:

- A 1km long road segment.
- Cars being introduced at a in-flow rate of 5.0 cars per second onto the road.

`(b) (i)`. Measure the flow rate for each of the three scenarios above. Do the results agree with your analytical results in `(a)`? Re-run the simulation with a lower in-flow rate of 1.0 car per second being introduced into the road. Are the relative flow rates of the three scenarios look the same as before? If not, explain why.

Problem 4: Shorest path vs quickest path

(a). Consider the following modelling scenario:



where:

- Pedestrians are introduced in the green area as source and the red area as destination.
- New pedestrians are introduced at a rate of one person per second.
- There is a barrier at y = 12m, with two openings:
  - The first 0.9m wide opening is located at x ∈ [3.0m, 3.9m].
  - The second opening is 2m wide and is located at the right-hand side of the room, i.e. x ∈ [18m, 20m].
- Pedestrians use the shortest path to get from source to destination.

(a) (i). Run the simulation for 90 seconds and find out how many people were evacuated within those 90 seconds.

(a) (ii). Modify the previous scenario, so that pedestrians use the *quickest* path, as opposed to the *shortest* path.

How many people were evacuated within 90 seconds when using the quickest path, as opposed to the shortest path?