PEDESTRIAN FLOW SIMULATION

DOCUMENTATION – A Crossing Flow of Pedestrians in a corridor

ENVIRONMENT:

Built with Linux:

Operating System - OpenSUSE Tumbleweed (x86-64)

Programming Language - Python 3 or higher

Additional Libraries Required:

- Numpy - for array operations
- Yaml - for reading input file
- Matplotlib - For plotting pedestrian positions at time T

RUNNING THE CODE:

1. Open linux terminal and enter the file location
2. The input variables for the simulation can be changed in the "input.yml" file present in the folder (For defaults skip this step)
3. Now execute the command in the linux terminal - *python pedsim.py* (or) *python3 pedsim.py*

```
arunaachalam@localhost:~/Documents/newDynamics> ls
collisionforcemodel  input.yml  pedsim.py
arunaachalam@localhost:~/Documents/newDynamics> python3 pedsim.py
```

Figure 1- Folder and Python Command to run the Code

INPUT FILE:

The following input parameters can be changed with "input.yml" file

- Mass of the pedestrian, pedestrian_mass
- Desired Velocity of pedestrian, desired_velocity
- Maximum Velocity of pedestrian, max_velocity
- Pedestrian flux at Entrance A, flux_up
- Pedestrian flux at Entrance C, flux_right
- Print Control for pedestrian position, print_png

Please note: Do not change other parameters. Temporary implementation and can cause errors

```
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# SIMULATION PARAMETERS THAT CAN BE CHANGED
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
spawn_method     : random         #point/random
pedestrian_mass : 75.0           #Mass of pedestrians (set at default 75 kg)
desired_velocity: 1.1            #Prefered Velocity of the Pedestrians [1.1, 1.3]
maximum_velocity: 1.1            #Maximum velocity of the Pedestrians (SET: 1.3)
flux_right       : 0.5          #Pedestrian flux at 'C' in interval [0.2,2.0]
flux_up          : 0.5           #Pedestrian flux at 'A' in interval [0.2,2.0]
print_png        : 0.5           #PRINT CONTROL in seconds
```

Figure 2 - Changeable Parameters in "input.yml" file

PROBLEM DEFINITION:

The geometry below represents two intersecting corridors with walls. Pedestrians with a constant flux Φ enter the corridor from the bottom (region A) and left (region C). The pedestrians entering from region A leave the corridor at region B and the pedestrians from region C leaves the corridor at region D. All dimensions are in meters.

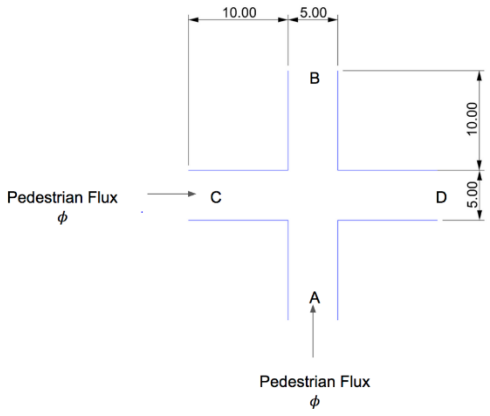


Figure 3 - Problem description with geometry

Set up a solver in the desired programming environment for simulating the pedestrian flow. Representation of individual pedestrians in simulation is necessary. Perform the simulation with at least four different pedestrian flux values in the range $\Phi = 0.2 - 2$ pedestrians/second.

For the representation of the pedestrians, assume that:

- They have a circular cross-section with radius r=0.25m
- They have a desired velocity in the range of $1.1 - 1.3$ m/s
- Their maximum acceleration is 2 m/s2

Integrate the motion of the pedestrians for 60 sec, considering the desired target path/exit and velocity, as well as the avoidance of other pedestrians.

FURTHER ASSUMPTIONS AND SIMPLIFICATIONS

Model used	: Social Force Model
Spawn Method	: Pedestrians spawn at a fixed Point or at random points
Simplified Wall	: Pedestrians take the shortest path to target, No deviations
Force Constants	: Obtained from Literatures
Other assumptions	: No rotational abnormality

SOCIAL FORCE MODEL

Helbing et al. (1995) proposed that the social behavior of the pedestrians in a crown can be described as social forces. Thus, a microscopic simulation of pedestrians can be performed. This model is widely used and tested. This model describes pedestrians in a crowd moving with a preferred velocity without collisions and is the best model to simulate the given scenario.

Preferred Force

The pedestrian tries to reach the desired target with a preferred velocity(v^p), taking the shortest possible path, when unhindered.

$$F_i^p = \frac{m}{\tau}(v_i^p - \dot{x}_\iota)$$

where,

m – mass of the pedestrians in kg, τ – relaxation time in s, v_i^p – preferred velocity of pedestrian (m/s), $\dot{x}_\iota$ – velocity of the pedestrian at given time (m/s)

Wall force

When the pedestrian tries to move near the wall, an exponential force is developed trying to push him/her away from the wall.

$$F_{iw}^w = F_0^w \hat{r}_{iw} e^{\frac{-|r_{iw}|}{\sigma_w}}$$

where,

F_0^w, σ_w – Model parameter for SFM model, $\hat{r}_{iw}$ – Unit vector between the wall and the pedestrian position, $|r_{iw}|$ – distance between the wall and the pedestrian

From Helbing et al. (1995) the modal parameters are chosen as, $F_0^w = 50\ m/s^2$ and $\sigma_w = 0.2\ m$

Social Repulsion Force

When the pedestrians are in range, they tend to repel each other thus generating a social repulsion force.

$$F_{ij}^s = F_0^s \hat{r}_{ij} e^{\frac{2R-|r_{ij}|}{\sigma_s}}$$

where,

F_0^s, σ_s – Modal parameters for SFM model, $\hat{r}_{ij}$ – Unit vector between affecting and affected pedestrians, R – Radius of the pedestrians in m, $|r_{ij}|$ – distance between the affecting and the affected pedestrians

From Helbing et al. (1995) the modal parameters are chosen as, $F_0^s = 2000\ N$ and $\sigma_s = 0.5\ s$

Physical Force

Hassan et al (2017) suggested that the physical force between affecting and affected pedestrians can be divided into pushing and friction forces. These come into effect when the pedestrians come closer.

$$F_{ij}^{Physical} = F_{ij}^{pushing} + F_{ij}^{friction}$$

$$F_{ij}^{pushing} = F_0^p \hat{r}_{ij}(2R - |r_{ij}|)$$

$$F_{ij}^{friction} = F_0^f \hat{r}_{ij}(2R - |r_{ij}|)\Delta v_{ji}\, \hat{n}_{ij}$$

The model parameters are chosen as $F_0^p = 12000\ N$ and $F_0^f = 24000\ N$

Equation of Motion

$$\dot{v}_i = \sum F_i$$

$$\dot{x}_i = |v_i| \max(v_i, v_{\max})$$

SIMPLIFIED ALGORITHM

Load Input file and set simulation variables

for *i < maximum iterations*:

    Spawn Pedestrians and update Pedestrian List

    Calculate Average Velocity of the world

    for *p* in *[list of pedestrians]*:

        Calculate Forces acting on pedestrian *[p]*

        Update Velocity for pedestrian *[p]*

        Update Position of the pedestrian *[p]*

    Eliminate Pedestrians who exited the world

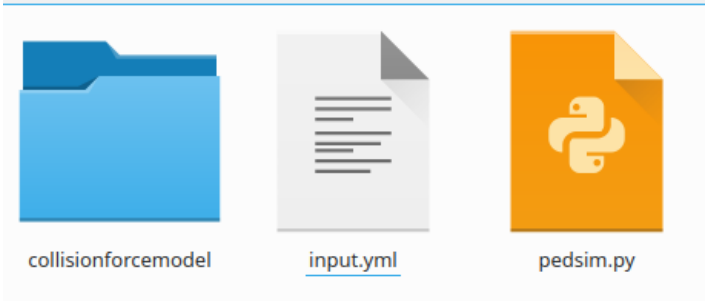
CODE IMPLEMENTATION

The code containing folder consists of a folder and two files:

input.yml : Simulation variables are given as input in this file

pedsim.py : Contains main() function, the Algorithm for simulation is written here

Folder : named (collisionforcemodel) contains associated



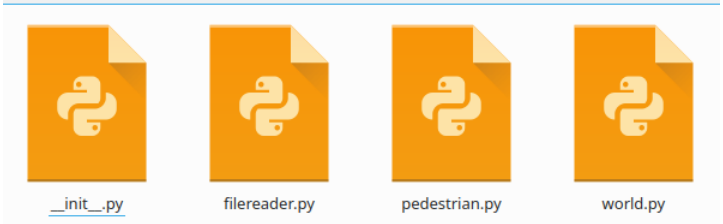
Files inside the folder collisionforce model:

__init__.py : Loads all modules in folder

pedestrian.py : Contains class Pedestrians

world.py : Contains the class World

filereader.py : Contains the class FileReader



RESULT VERIFICATION (SEE PowerPoint for Results):

As I could not find any specific literature to verify the code, I tried to verify it logically.

Case – 1

Consider pedestrian entering at only one side of the geometry. Under Low pedestrian flux, there should be no congestion and thus two conditions should be true

1. All pedestrians should enter at Side ‘C’ and leave at side ‘D’
2. The average velocity of the world should be the Preferred Velocity with minor fluctuations

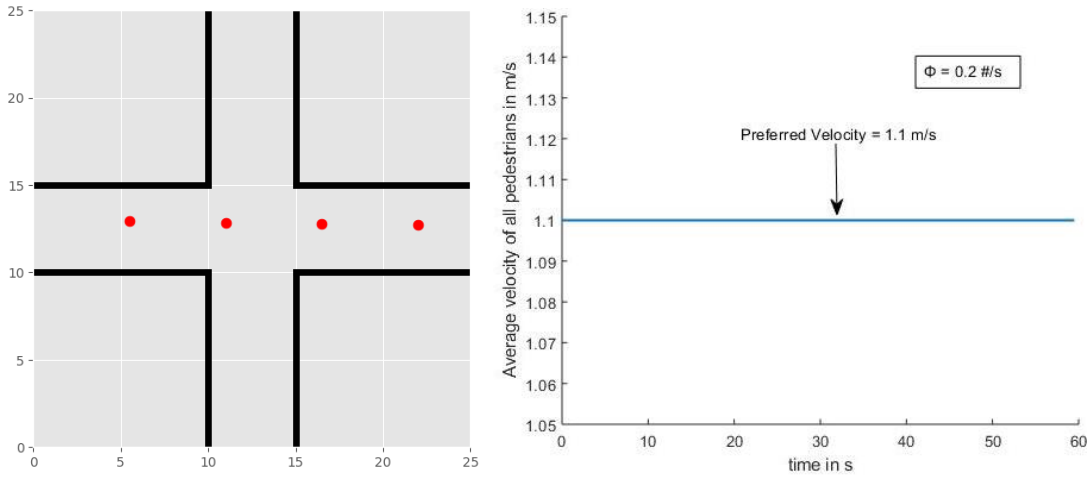


Figure 4 - At low pedestrian flux Pedestrians travel from left to right with preferred velocity

Case – 2

Consider Pedestrians entering at only one side of the geometry. Under High pedestrian flux, there should be repulsion force, resulting in congestion and thus two conditions should be true

1. Pedestrians try to spread from each other when entering from one point and form lane like structures
2. The average velocity of the world should reduce due to congestion

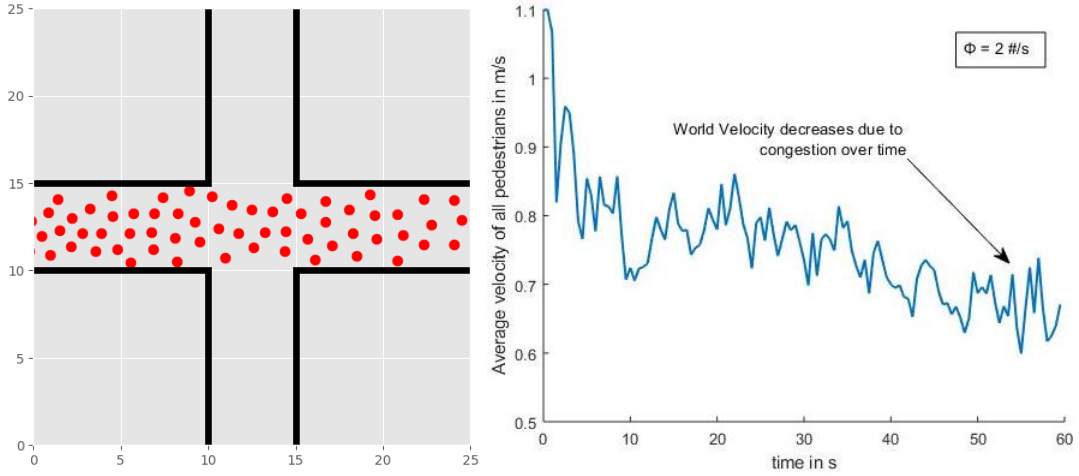


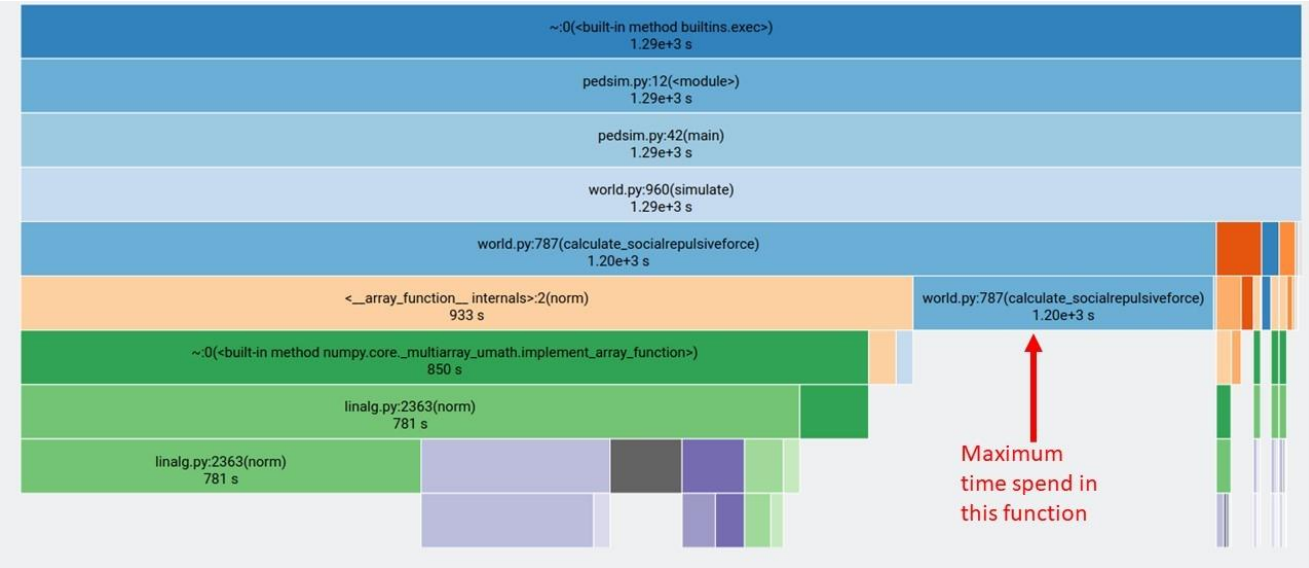
Figure 5 - At high pedestrian flux congestion leads to reduced world velocity

FOR RESULTS AND SUMMARY SEE THE ACCOMPANYING POWER POINT FILE IN THE REPOSITORY

PERFORMANCE METRICS:

With the help of cProfil and snakeviz, the program is profiled and visualized. On observation, as expected the maximum program time is spent on calculating repulsive force.

In Class World: calculate_social_repulsion_force (args)



The reason is the nested for loop structure. But this time can be reduced by parallelizing the loop

```
Parallel for p in list_of_pedestrians:
        Parallel for r in list_of_pedestrians
            Calculate Social Repulsion Force
```

As each pedestrian have its own characteristics, parallelization does not depend on other calculations and the code is becomes highly scalable.

NOTE: the memory access to be improved with updated algorithm has in the future.