SIM507 · WEEK 2 · PLAIN-LANGUAGE EDITION

Pedestrian dynamics on the platform

Two programs you never touch directly: one walks a simulated crowd, the other measures it. The whole week is about the single file that passes between them — what it must contain, and how to check it is honest.

Jakub Dubec · jakub.dubec@stuba.sk · FIIT STU

Where we are

LAST WEEK ESTABLISHED

A simulation produces output files, and each file comes with a published description of what it should contain. That published description is the promise that makes a result re-runnable.

THIS WEEK MAKES IT CONCRETE

We meet the first real pair of programs and the one file they pass between them — and we learn to check that file against its promise.

Last week: the idea of a contract. This week: a real contract you can touch.

CRD503 teaches the crowd physics.

SIM507 teaches how you drive the tools.

A sister course explains *why* a crowd slows down when it gets dense. Here we treat both programs as sealed boxes. Our skill is operating them and checking the file that joins them — not deriving the physics.

First, the words

Seven terms. Learn these and the rest of the week is easy.

The words you need

Simulator
A program that imitates a real physical process on a computer.

Trajectory
The path each simulated person walked — a list of positions over time.

Output file
Any file a simulator produces when it finishes.

Pipeline diagram
A flow chart where work only moves forward, step to step — no loops back.

Docker image
A sealed software package with everything the program needs to run — like a recipe box that ships its own ingredients.

Parquet file
A storage file like a spreadsheet, but strict about the type of each column.

Published description
The program's own list of the files it expects in and produces out, with the type of every column.

Each of these comes back, in context, later — you don't have to memorise them now.

Two everyday anchors

📦 A **Docker image** is a sealed lunchbox: it carries its own program, its own libraries, its own settings. Hand it to any computer and it runs the same way — no "works on my machine".

🗂️ A **Parquet file** is a spreadsheet with the column types locked. A plain CSV lets you type the word "twelve" into a number column; Parquet refuses — every column has a declared type and the file enforces it.

Sealed box, strict spreadsheet. Hold both — the week runs on them.

Act I

Two programs, one file between them

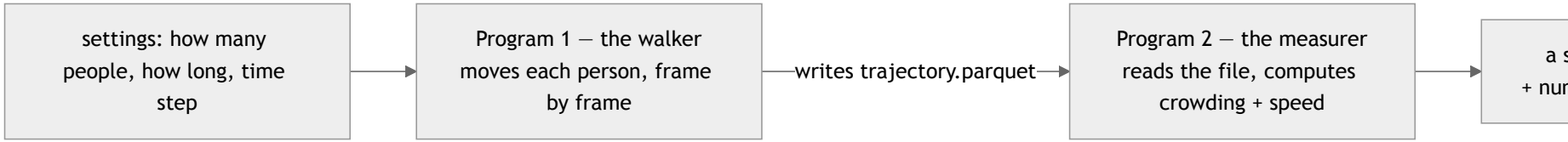
Two specialists, one handshake

Picture a lab with two technicians who never meet.

> 🤝 One technician **runs the experiment** and writes every result onto a **standard form**. A second technician picks up that form and **analyses it**. They never speak, never share a desk. They agree on exactly one thing: the **layout of the form**. Get the form right and either technician can be swapped out without telling the other.

That standard form is the file `trajectory.parquet`.

The same idea, as a diagram



- **The walker** moves pedestrians and records where everyone was each frame
- **The measurer** moves no one — it *measures* a path that already exists
- They never share memory. They agree on **one file**.

> **Pipeline diagram** — work flows left to right only. The walker's output becomes the measurer's input; nothing loops back.

Act II

The file they agree on: `trajectory.parquet`

Five columns, one row per person per frame

> **Schema** — the locked list of columns in the file: each column's name, its type, and what it means.

Column	Type	Plain meaning
`frame`	int32	Which frame this is: 0, 1, 2, …
`t_s`	float64	The clock time of that frame, in seconds
`agent_id`	int32	Which simulated person this row is about
`x`	float64	Position along the room’s long side, in metres
`y`	float64	Position along the room’s short side, in metres

One row = one person at one instant.

The same file, two jobs

The walker and the measurer describe this file differently — on purpose.

THE WALKER SAYS

"This is my **main output**." It calls the file's job **trajectory** — the thing it exists to produce.

THE MEASURER SAYS

"This is something I **need but did not make**." It calls the file's job **incoming input**, and records who produced it.

🔌 Same as a **plug and a socket**: the appliance maker and the wall-socket maker never meet, but both build to the same plug shape. The file's "job" label is that agreed shape.

How big is this file?

A little arithmetic, one step at a time.

1. **How often?** The walker records a frame every **0.05 seconds** — that is 20 frames every second.
2. **For how long?** A run lasts **180 seconds**. So 180 ÷ 0.05 = **3 600 frames**.
3. **How many people?** Say **16**. Every person gets one row per frame.
4. **Multiply.** 3 600 frames × 16 people = **57 600 rows**.

A 3-minute run with 16 people = 57 600 rows.

A contract is only useful if it is checked

> Think of a **customs officer** who reads the label on a parcel. Wrong label → the parcel is **refused at the border**. The officer does not guess what's inside and wave it through.

- Rename `t_s` to `time`? The measurer sees a plausible-but-wrong file.
- Store `x` as a less precise number type? Silent precision loss.
- **Correct behaviour: refuse the file** — never emit a confident wrong answer.

A scientific instrument that fails *silently* is the worst kind of failure.

Act III

What the measurer produces

Crowding, measured per time window

Per-window summary — one row for each short slice of time, instead of one row per person per frame.

Column	Plain meaning
`t_s`	When this time window starts, in seconds
`mean_density_ped_m2`	Average crowding in that window, in people per square metre
`mean_speed_m_s`	Average walking speed in that window, in metres per second
`n_active`	How many people were inside the measured area in that window

people per square metre (ped/m²) — the natural crowding unit. About 0.5 = a sparse room; 4 and above = a dangerous jam.

How "crowding" is measured: each person owns a patch

🗺️ Picture the floor split into **territories**: every spot belongs to whoever is *nearest*. Each person "owns" the patch of floor closest to them. A person hemmed in by neighbours owns a **small** patch → high crowding. An isolated person owns a **large** patch → low crowding.

Voronoi method — the formal name for that "nearest-owner" split of the floor. Each person's local crowding is one divided by the area of their patch.

Small patch = crowded · big patch = sparse.

The workbook’s offline stand-in

The real measurer needs a library we can't install offline, so the workbook estimates crowding a simpler way.

Draw a circle from each person out to their **nearest neighbour**; the disc halfway between them is roughly the space that person "owns". Smaller circle → more crowded.

$$\hat{\rho}_i = \frac{4}{\pi d_i^2}$$

$\hat{\rho}_i$ = estimated crowding for person i · d_i = distance to their nearest neighbour · the rest just turns that distance into the area of the little disc. Closer neighbour → bigger number → more crowded.

This reads *local* spacing, so it runs higher than the room-average. It recovers the right **ballpark**, not the exact number. The lesson is the **shape and the schema**, not the precise value.

Why throw away the start of each run?

> 🍳 Like a frying pan: the first minute is just **warming up**, not cooking. You don't judge the heat until the pan settles.

- The people start from an **artificial starting layout**
- They need time to settle into steady, natural motion
- Averaging over that warm-up would **bias** the crowding number

THE RULE THE LAB USES

Throw away the **first 17%** of every run — about the first 30 seconds of a 3-minute run — before computing any average.

This rule lives in the run's **settings**, written down and versioned — not hidden inside code. Visible settings are what make a result reproducible.

Act IV

The one chart the whole pipeline exists for

Three runs, three points

The lab ran the pair three times in an 8 m × 12 m seminar room (≈ 96 m^2 of walkable floor), changing only how many people walked.

Run	People	Crowding (ped/m^2)	Speed (m/s)
Run A	4	**0.042**	**1.169**
Run B	16	**0.167**	**1.139**
Run C	48	**0.500**	**0.708**

More people → slower walking. Every single time.

"Every time" has a name: rank correlation

> **Rank correlation** — a score from −1 to +1 asking only: as one thing goes up, does the other go reliably down (or up)? +1 = always up together; −1 = always opposite; 0 = no pattern.

📉 Like asking "does the queue get slower the more people join it?" If it slows **every single time**, with no exception, the score is a perfect −1.

Our three points score −1.0: speed fell every time crowding rose.

The illustrative curve (the data are sovereign)

We draw a smooth curve through the three points *for the eye only* — the standard Weidmann speed–density law.

🚶 Walking speed depends on the space each person has: lots of space → people walk near their free speed; almost none → they grind to a halt.

$$v(\varrho) = v_0 \, [\, 1 - \exp(-\gamma(\tfrac{1}{\varrho} - \tfrac{1}{\varrho_{\max}}))]$$

Free-flow speed v_0, knocked down by an exponential that depends on the space per person (1/ρ).

In plain terms: **v_0** is the free-flow walking speed (how fast you go with the floor to yourself), and **γ** controls how sharply speed drops as the crowd thickens. **ρ** is crowding; **ρ_max** is the crowding at a total jam.

Three points cannot prove a curve. The lab reports the *points*; the curve is a teaching aid for the eye, not evidence.

What is a "regime"?

> **Regime** — a band of crowding where the crowd behaves in one characteristic way. The measurer sorts crowding into four bands.

Free-flow
below 0.5 ped/m^2 — walk at your own pace.

Light congestion
0.5–2.0 ped/m^2 — you start adjusting to others.

Dense flow
2.0–4.0 ped/m^2 — movement is shaped by the crowd.

Jam
4.0 ped/m^2 and up — shuffling, dangerous.

2 of 4

regimes reached

A 96 m^2 room with these people-counts reaches free-flow and light congestion only.
A real jam needs roughly 400 people or a narrow choke-point — deferred to a follow-on study.

Act V

Operating the pair on the platform

Launch a run

> **Run** — one single execution of the pair with one set of settings.

```
launch_run(
    simulation="exp-s1",
    settings={"people": 16},
    your_session="<your-session-id>",
)
```

- You change only **how many people**; the room, the length, the time step and the discard rule are filled in from the shared settings for you
- The launch hands back a **ticket number**; you check on it until it reports **finished**
- The whole sweep of people-counts can run automatically, or you can watch it step by step

Inspect the outputs, find the record

> **Signed link** — a temporary download link to a file in cloud storage that expires after a short time, so it can't be shared forever.

The headline numbers + how the run was made:

```
read_run(run_id)   # the run's settings, versions, and summary numbers
```

The raw data file (via a short-lived link):

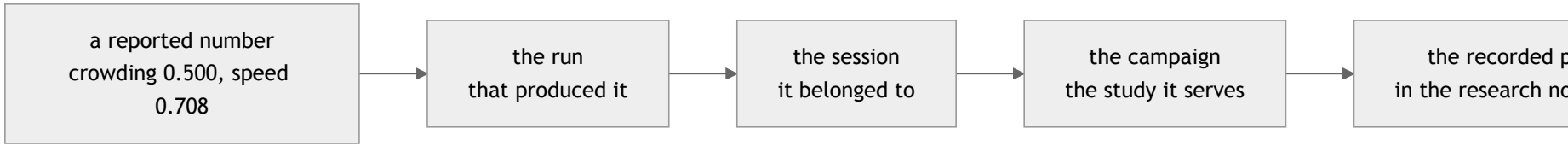
```
download_link(run_id, "voronoi_summary")   # expires in minutes
```

Where it lands in the research notebook:

- a generated page for the run, under the experiments section
- a page for the whole campaign, tracking its latest batch

Every number has a paper trail

> **Run · session · campaign** — three nested levels: a **run** is one execution; a **session** is a batch of runs done together; a **campaign** is the whole study the sessions belong to.



Any number you cite must walk this chain back to its source. No orphan figures.

The same `trajectory.parquet` is Week 3's input.

Next week swaps the measurer for a radio-channel simulator. It lists this same file as its **incoming input** — the crowd you walked this week becomes the crowd that next week's radio waves must pass through.

Recap — back to our five goals

- The simulation is **two programs joined by one file** — the walker and the measurer
- That file has a **locked schema**: `frame, t_s, agent_id, x, y` — **check it at the edge**
- The measurer turns it into a **per-window summary**; throw away the warm-up before averaging
- Three real runs: more people → slower walking, a perfect **−1.0** rank correlation, two regimes only
- **Launch → inspect → find the record**, and trace every number back to its run

One file, checked at the edge, traced to its source. That is the skill.

Further reading

The walker (JuPedSim runner)
The curated note on the pedestrian-walking program: its contract, its inputs and outputs, and its parameter reference.

The measurer (PedPy analyser)
The curated note on the measurement program: the crowding methods, the speed methods, and the regime bands.

Liddle et al., 2022
The empirical study the regime bands derive from — the source of the free-flow and light-congestion ranges.

EXP-S1 simulation sandbox
The vault note that chains these two programs, including the Geometry A room definition.

Read the Contract / Inputs / Outputs sections of the two program notes for operation; the rest is upstream reference.

Week 3 → radio-channel simulation

Reuse this week's `trajectory.parquet` as the crowd a radio simulator must see through.