TransitLab SimMETRO

Mojtaba Yousefi

Aug 28, 2024

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Project Overview

TransitLab SimMETRO is a sophisticated simulation tool developed upon decades of research (Koutsopoulos and Wang [KW07] and Zhou [Zho22]) and development to address the operational challenges faced by heavy rail systems in major metropolitan areas, particularly during peak periods when demand is highest. This tool provide microscopic simulation model for heavy rail systems, enabling detailed analysis of train behaviors, signaling systems (fixed-block and moving block), and interactions between trains and passengers. By employing TransitLab SimMETRO, reseachers can evaluate various operating strategies such as skip-stop, station consolidation, and dwell control to mitigate capacity bottlenecks.

The simulation framework is built using Python in a unix environment, ensuring robust performance and flexibility. The model's accuracy and reliability can be further enhanced through a calibration process using data from Operational Control Systems [WK11]. Numerous visualization tools allow users to analyze simulation outputs.

Numerous case studies (see Zhou *et al.* [ZKS20] and Zhou and Koutsopoulos [ZK22]) demonstrate how MIT TransitLab SimMETRO has helped agencies address congestion, improve service reliability, and support long-term planning decisions.

1.2 Features

1. Detailed modeling of signal systems, including both fixed-block and moving-block, with precise train movement simulations at the individual vehicle level.
2. Accurate dwell time modeling at stations to reflect real-world scenarios.
3. Realistic passenger boarding and alighting behaviors.
4. Implementation of headway-based dispatching to model stochasticities in rail operations.
5. Comprehensive simulation outputs for thorough analysis.
6. Multiple visualization tools available for detailed examination of simulation data.
7. Customizable train movement models to suit various rail systems.

1.3 License

The project License is yet to be determined. Please contact the development team for more information.

CHAPTER

TWO

SETUP GUIDE

This project uses Pipenv for dependency management and packaging. You can learn about the dependencies looking at the Pipfile at the project root directory. Assuming you have a working version of pip, you can install Pipenv using pip:

```
$ pip install --user pipenv
```

Pipenv will take care of installing dependencies and creating the virtual environment. Run the following command from the project root directory (where the Pipfile and Pipfile.lock are located):

```
$ pipenv install
```

You can activate the virtual environment using:

```
$ pipenv shell
```

Depending on Python versions available on your system you may get an error like:

```
Error: the specified Python version (3.8) is not available on your system.
```

It is recommended to install the required Python version using pyenv. You can install pyenv using the instructions at https://github.com/pyenv/pyenv?tab=readme-ov-file#installation.

You may need to reactivate your shell after installing pyenv/pipenv to make sure they are added to the PATH.

Once you have pyenv installed, Pipenv will ask you if you like to use pyenv to install the required Python version. You can say yes and Pipenv will install the required Python version in the virtual environment.

CHAPTER

THREE

GETTING STARTED

3.1 Quick Start

1. Prepare the input files:
    - **Ensure you have the following files in the *inputs* directory:**
        - infra.json: Infrastructure data
        - slow_zones.json: Slow zone definitions
        - demand/odx_demand.csv: Passenger demand data
        - schedules/empirical_schedule.json: Train schedule data
2. Configure the simulation:
    - Open the *load-balance/config.yaml* file
    - **Adjust the following parameters as needed:**
        - simulation.number_of_replications
        - simulation.start_time_of_day
        - simulation.end_time_of_day
        - demand_level
        - station
        - short_turning
        - logger.should_log_trajectories
        - logger.log_interval
3. Run the simulation:

```
python -m transit_lab_simmetro.simulation_runner.runner
```

4. Check the output:
    - Look for the generated log files in the specified log folder (default: *log_folder_path*)
    - You'll find CSV and JSON files with various simulation data

3.2 Understanding the Inputs and Config

1. Infrastructure (infra.json):
    - Defines blocks, stations, and path information for Northbound and Southbound directions
2. Slow Zones (slow_zones.json):
    - Specifies areas with reduced speed limits
3. Demand Data (odx_demand.csv):
    - Contains passenger arrival rates at 15-minute intervals for weekdays and non-weekdays for each OD pair
4. Schedule Data (empirical_schedule.json):
    - Defines train dispatch times and routes
5. Configuration (config.yaml):
    - simulation: Set replication count, start and end times
    - demand_level: Adjust overall passenger demand. Used as a multiplier to uniformly scale demand.
    - station: Specify holding station (e.g., "O-Hare" or "Clark/Lake")
    - short_turning: Set short turning location ("UIC" or "Western")
    - logger: Configure logging options

By adjusting these inputs and configurations, you can simulate various scenarios and analyze the rail system's performance under different conditions.

CHAPTER

FOUR

USAGE

4.1 Basic Usage

Explain how to use the project with simple examples.

4.2 Advanced Usage

Dive into more complex use cases and features.

CHAPTER

FIVE

CONFIGURATION

5.1 Configuration Options

Document configuration options, environment variables, etc.

5.2 Default Configuration

Explain the default setup and how to customize it.

CHAPTER

SIX

ARCHITECTURE

6.1 Components Overview

6.1.1 Infrastructure Module

Infrastructure components for transit system simulation.

This module provides essential classes and types for modeling the physical infrastructure of a transit system, including blocks, paths, stations, and control systems.

Classes:

Block: Represents a fixed section of track. Terminal: Special block type for end-of-line stations. MovingBlock: Implements moving block signaling system. Path: Defines a sequence of blocks forming a route. Station: Represents a passenger boarding and alighting point. SignalControlCenter: Manages traditional fixed-block signaling. MovingBlockControl: Manages moving block signaling system.

Types:

BlockType: Union type for different block implementations.

The module supports both fixed-block and moving-block signaling systems, allowing for flexible and realistic transit network modeling.

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**AbstractBlock**

Bases: ABC

__init__(*block_id: str*, *block_alt_name: str*, *visible_distance: float*, *length: float*, *default_speed_code: float*, *station:* Station | *None* = *None*)

abstract current_speed_code(*requesting_train:* Train) → float

abstract property is_occupied: bool

abstract is_occupied_by(*train:* Train) → bool

set_slow_zone(*speed_limit: float*)

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**Block**

Bases: *AbstractBlock*

__init__(**args*, *speed_codes_to_communicate: Dict[str, float]* | *None* = *None*, ***kwargs*)

activate(*entering_train:* Train) → None

add_communicated_speed_code(*block_id: str*, *speed_code: float*) → None

add_observer(*observer:* SignalControlCenter) → None

block_logger: BlockActivationLogger | None = None

current_speed_code(*requesting_train:* Train) → float

deactivate(*exiting_train:* Train) → None

property id_of_last_train: str | None

property is_occupied: bool

is_occupied_by(*train:* Train) → bool

property last_train_visit_time: float

notify_observers() → None

remove_communicated_speed_code(*block_id: str*) → None

set_slow_zone(*speed_limit: float*)

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**DispatchingBlockDecorator**

Bases: *Block*

__init__(*block:* Block, *path:* Path, *dispatch_margin: float = 120*, *upstream_blocks: List[str] = []*)

activate(*entering_train:* Train) → None

is_it_clear_to_dispatch() → bool

property last_train_visit_time: float

ready_to_dispatch(*train:* Train) → bool

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**DispatchingMovingBlockDecorator**

Bases: *MovingBlock*

__init__(*block:* MovingBlock, *path:* Path, *dispatch_margin: float = 120*, *upstream_blocks: List[str] = []*)

activate(*entering_train:* Train) → None

is_it_clear_to_dispatch() → bool

property last_train_visit_time: float

ready_to_dispatch(*train:* Train) → bool

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**MovingBlock**

Bases: *AbstractBlock*

__init__(**args*, ***kwargs*)

activate(*entering_train:* Train) → None

property civil_speed_limit: float

current_speed_code(*requesting_train:* Train) → float

property current_train: *Train* | None

deactivate(*exiting_train:* Train) → None

property is_occupied: bool

is_occupied_by(*train:* Train) → bool

next_train(*requesting_train:* Train) → *Train* | None

register_moving_block_control_center(*moving_block_control*) → None

property sorted_current_train_list: List[*Train*]

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**MovingBlockControl**

Bases: object

__init__(*blocks: List[*MovingBlock*]*, *safety_margin: float = 200*) → None

get_distance_to_next_train(*asking_train:* Train) → float

get_speed_code(*asking_train:* Train, *asking_block:* MovingBlock) → float

needed_braking_distance(*asking_train:* Train) → float

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**OffScanSymptomaticBlockDecorator**

Bases: *Block*

__init__(*block:* Block, *path:* Path, *offscan_probability: float = 0.01*)

activate(*entering_train:* Train) → None

current_speed_code(*requesting_train:* Train) → float

set_unsymptomatic() → None

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**Path**

Bases: object

__init__(*direction: str*, *blocks: List[*MovingBlock | Block*]*, *slow_zones: List[*SlowZone*] = []*)

copy()

get_all_stops_ahead(*block_index: int*) → List[*Station*]

get_all_stops_ahead_which_are_served(*block_index: int*) → List[*Station*]

get_block_by_id(*block_id: str*) → *MovingBlock* | *Block*

get_block_index_by_id(*block_id: str*) → int

get_distance_to_the_next_station(*current_block_index: int*, *current_location_on_block: float*) → float

get_next_station(*current_block_index: int*) → *Station* | None

get_next_train(*current_block_index: int*) → *Train* | None

get_previous_train(*current_block_index: int*) → *Train* | None

get_total_length() → float

get_total_travelled_distance(*current_block_index: int*, *distance_travelled_in_current_block: float*) → float

is_inspected() → bool

is_short_turn() → bool

is_short_turned_at_this_station(*station:* Station) → bool

make_dispatching_block(*block_id: str*, *dispatch_margin: float = 120*, *upstream_blocks: List[str] = []*)

make_offscan_block(*block_index: int*, *offscan_probability: float*)

path_distance_alignment() → None

set_block_slow_zone(*block_id: str*, *speed_limit: float*)

short_turn(*train:* Train)

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**SignalControlCenter**

Bases: object

__init__(*blocks: List[*Block*]*)

get_block_by_id(*block_id: str*) → *Block* | None

restore_speed_codes_to_upstream_blocks(*block:* Block) → None

send_speed_codes_to_upstream_blocks(*block:* Block) → None

update(*block:* Block) → None

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**SlowZone**

Bases: object

__init__(*block_id: str*, *reduced_speed_limit: float*)

classmethod from_dict(*data: dict*) → *SlowZone*

to_dict() → dict

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**Station**

Bases: object

__init__(*station_name: str*, *location_relative_to_block: float*, *direction: str*, *arrival_rates:* ArrivalRate)

board_passengers_based_on_destiations_and_probability(*train_capacity: int*, *served_destinations: List[str]*, *probability_of_boarding_any_train: float*) → List[*Passenger*]

board_passengers_onto_train(*train_capacity: int*, *served_destinations: List[str]*) → List[*Passenger*]

generate_and_add_passengers(*train:* Train, *holding_time: float = 0*) → None

get_dwell_time(*door_metrics: List[Tuple(int, int, int)]*) → float

property last_train_visit_time: float

simulation: *Simulation*

station_logger: StationLogger | None = None

class transit_lab_simmetro.simulation_engine.infrastructure.__init__.**Terminal**

Bases: *Block*

__init__()

activate(*entering_train:* Train) → None

current_speed_code(*requesting_train:* Train) → float

6.1.2 Passenger Module

This module imports and exposes key components for passenger modeling:

Classes:

Passenger: Represents an individual transit user. ArrivalRate: Models passenger arrival patterns at stations. PassengerGenerator: Creates passenger objects based on arrival rates.

These components form the foundation for simulating passenger behavior within the transit system.

class transit_lab_simmetro.simulation_engine.passenger.__init__.**ArrivalRate**

Bases: object

Calculates the arrival rate and flow probability for a given station, time, and direction in a transit system.

The *get_lambda_for_station* method calculates the total arrival rate for a given station, time, and direction by summing the smoothed arrival rates for all destination stops from that station.

The *get_p_for_station* method calculates the flow probability for a given station, time, and direction by dividing the through-flow rate (sum of arrival rates to all downstream stations) by the total arrival rate at that station.

The *get_a_i* method calculates the cumulative flow probability from a start station to a critical station in a given direction. This is used to compute the *get_lambda_bar* method, which calculates the average arrival rate between a start station and a critical station in a given direction.

__init__(*filename*, *demand_factor: int = 1*)

get_a_i(*current_hour: float*, *current_weekday: bool*, *start_station: str*, *critical_station: str*, *direction: str*) → float

Calculates the probability (a_i) of passengers arriving at each station between the start and critical stations in a given direction.

The probability is calculated by: 1. Sorting the stations between the start and critical stations in the given direction. 2. Iterating through the stations between the start and critical stations. 3. For each station, calculating the probability (p_k) of passengers passing through that station. 4. Multiplying the probabilities (p_k) to get the overall probability (a_i).

Parameters

- **current_hour** (*float*) – The current hour of the day.
- **current_weekday** (*bool*) – Whether the current day is a weekday or not.
- **start_station** (*str*) – The start station.
- **critical_station** (*str*) – The critical station.
- **direction** (*str*) – The direction of travel.

Returns
The probability (a_i) of passengers arriving at each station between the start and critical stations.

Return type
float

`get_all_destination_stops_for_origin`(*origin_stop: str*) → List[str]

Gets all the destination stops that can be reached from the given origin stop.

Parameters
origin_stop (*str*) – The origin stop to get the destination stops for.

Returns
A list of all the destination stops that can be reached from the given origin stop.

Return type
List[str]

`get_all_destination_stops_for_origin_and_direction`(*origin_stop*, *direction*) → List[str]

Gets all the destination stops for a given origin stop and direction.

Parameters

- **origin_stop** (*str*) – The origin stop to get the destination stops for.
- **direction** (*str*) – The direction of travel.

Returns
A list of all the destination stops that can be reached from the given origin stop in the specified direction.

Return type
List[str]

`get_all_previous_stops_for_station_and_direction`(*station: str*, *direction: str*) → List[str]

Gets all the previous stops for a given station and direction.

Parameters

- **station** (*str*) – The station to get the previous stops for.
- **direction** (*str*) – The direction of travel.

Returns
A list of all the previous stops that come before the given station in the specified direction.

Return type
List[str]

`get_lambda_bar`(*current_hour: float*, *current_weekday: bool*, *start_station: str*, *critical_station: str*, *direction: str*) → float

Calculates the average arrival rate (lambda_bar) for passengers between a start station and a critical station in a given direction.

The arrival rate is calculated by: 1. Sorting the stations between the start and critical stations in the given direction. 2. Calculating the probability (a_i) of passengers arriving at each station between the start and critical stations. 3. Calculating the arrival rate (lambda_i) for each station between the start and critical stations. 4. Summing the product of a_i and lambda_i for each station between the start and critical stations. 5. Adding the arrival rate for the critical station.

Parameters

- **current_hour** (*float*) – The current hour of the day.
- **current_weekday** (*bool*) – Whether the current day is a weekday or not.
- **start_station** (*str*) – The start station.
- **critical_station** (*str*) – The critical station.
- **direction** (*str*) – The direction of travel.

Returns
The average arrival rate (lambda_bar) between the start and critical stations.

Return type
float

get_lambda_for_station(*current_hour: float*, *current_weekday: bool*, *station: str*, *direction: str*) → float

Calculates the total arrival rate (lambda_i) for a given station and direction.

The total arrival rate is calculated by summing the smoothed arrival rates for all the destination stops that can be reached from the given station in the specified direction.

Parameters

- **current_hour** (*float*) – The current hour of the day.
- **current_weekday** (*bool*) – Whether the current day is a weekday or not.
- **station** (*str*) – The station to calculate the total arrival rate for.
- **direction** (*str*) – The direction of travel.

Returns
The total arrival rate (lambda_i) for the given station and direction.

Return type
float

get_p_for_station(*current_hour: float*, *current_weekday: bool*, *station: str*, *direction: str*) → float

Calculates the probability (p_k) of passengers passing through a given station in a specific direction.

The probability is calculated by: 1. Getting all the previous stations and next stations for the given station and direction. 2. Calculating the total arrival rate for all the previous stations to the given station, and the next stations. 3. Calculating the "through rate" which is the arrival rate for all the next stations. 4. Returning the ratio of the through rate to the total rate as the probability (p_k).

Parameters

- **current_hour** (*float*) – The current hour of the day.
- **current_weekday** (*bool*) – Whether the current day is a weekday or not.
- **station** (*str*) – The station to calculate the probability for.
- **direction** (*str*) – The direction of travel.

Returns
The probability (p_k) of passengers passing through the given station in the specified direction.

Return type
float

get_smoothed_rate(*current_hour*, *current_weekday*, *origin_stop*, *destination_stop*)

Gets the smoothed arrival rate for a given origin stop, destination stop, current hour, and current weekday.

Parameters

- **current_hour** (*float*) – The current hour of the day.
- **current_weekday** (*bool*) – True if the current day is a weekday, False if it's a weekend.
- **origin_stop** (*str*) – The origin stop.
- **destination_stop** (*str*) – The destination stop.

Returns

The smoothed arrival rate for the given parameters, or 0 if no matching entry is found.

Return type

float

is_southbound_trip(*origin_stop*, *destination_stop*)

sort_stations_by_direction(*direction*) → List[str]

class transit_lab_simmetro.simulation_engine.passenger.__init__.**Passenger**

Bases: object

Represents a passenger in the simulation. Passengers have various attributes such as arrival time, origin, direction, destination, boarding time, alighting time, waiting time, and travel time. The *Passenger* class also keeps track of the number of times a passenger has been denied boarding.

The *Passenger* class is ordered based on the passenger's arrival time, allowing for easy sorting and comparison of passengers.

The *passenger_logger* attribute is used to log passenger events, such as when a passenger alights from the train. The *set_not_loggable()* method can be used to disable logging for a specific passenger.

__init__(*arrival_time: float*, *origin: str*, *direction: str*, *destination: str*)

property alighting_time

property boarding_time

denied_boarding()

property journey_time

passenger_logger: PassengerLogger | None = None

set_not_loggable()

property travel_time

property waiting_time

class transit_lab_simmetro.simulation_engine.passenger.__init__.**PassengerGenerator**

Bases: object

__init__(*arrival_rate:* ArrivalRate)

generate_passengers(*current_hour: float*, *current_weekday: bool*, *boarding_stop: str*, *alighting_stop: str*, *delta_t_in_seconds: float*) → list

Generates a list of passenger arrival times based on the given parameters.

Parameters

- **current_hour** (*float*) – The current hour of the day.
- **current_weekday** (*bool*) – Whether the current day is a weekday or not.
- **boarding_stop** (*str*) – The boarding stop for the passengers.
- **alighting_stop** (*str*) – The alighting stop for the passengers.
- **delta_t_in_seconds** (*float*) – The time interval in seconds for which to generate passenger arrivals.

Returns
A list of passenger arrival times in seconds.

Return type
list

6.1.3 Simulation Module

The *Simulation* class is responsible for managing the simulation of a transit system. It handles the scheduling and updating of trains, as well as the overall simulation time and state.

The *SimulationContext* class is a context manager that sets up the necessary simulation-related attributes on the *Train*, *Passenger*, and *Station* classes when entering the context, and cleans up those attributes when exiting the context.

class transit_lab_simmetro.simulation_engine.simulation.simulation.**Simulation**

Bases: object

The *Simulation* class is responsible for managing the simulation of a transit system. It handles the scheduling and updating of trains, as well as the overall simulation time and state.

The *Simulation* class initializes with a schedule, path information, a signal control center, a train speed regulator, and various simulation parameters. It creates and manages a list of trains, dispatching new trains as scheduled and updating the existing trains at each time step. The class also provides methods to check the current simulation time and whether it is a weekday.

__init__(*schedule: Schedule | BaseSchedule*, *path: Dict[str, Path]*, *signal_control_center: MovingBlockControl | SignalControlCenter*, *train_speed_regulator: str*, *time_step: float = 0.5*, *start_hour: float = 5.0*, *is_weekday: bool = True*, *total_time: float = 14400*)

get_current_hour() → float

is_weekday() → bool

remove_train(*train: Train*) → None

run() → None

simulation_logger: SimulationLogger | None = None

class transit_lab_simmetro.simulation_engine.simulation.simulation.**SimulationContext**

Bases: object

A context manager that sets up the simulation context for various simulation entities.

When entering the context, it sets the *simulation* attribute on the *Train*, *DummyTrainDecorator*, *Passenger*, and *Station* classes, and resets the *_last_id* attributes on the *Train* and *Passenger* classes.

When exiting the context, it sets the *simulation* attribute on the *Train*, *Passenger*, and *Station* classes back to *None*.

__init__(*simulation:* Simulation)

6.1.4 Train Module

Train module for simulating rail vehicle behavior in a transit system.

This module provides classes and utilities for modeling train operations, including speed regulation, passenger management, and train state handling.

Classes:

Train: Represents a rail vehicle with its physical and operational properties. DummyTrain: A simplified train model for testing and special scenarios. DummyTrainDecorator: Decorator for creating dummy train instances. TrainSpeedRegulator: Manages train speed and acceleration based on signal system. TrainSpeedRegulatorCTA: CTA-specific implementation of speed regulation. TrainPassengerManager: Handles passenger boarding, alighting, and capacity management.

The module integrates various aspects of train operations: - Speed control and acceleration management - Passenger flow and capacity handling - Train state transitions (e.g., dwelling, accelerating, braking) - Support for different signaling systems (fixed-block and moving-block)

Key features: - Realistic acceleration and deceleration profiles - Passenger boarding and alighting simulations - Integration with infrastructure and signaling systems - Support for various operational scenarios (normal running, emergency braking, etc.)

The train module is central to the transit simulation, providing a detailed representation of train behavior and its interaction with passengers and infrastructure.

class transit_lab_simmetro.simulation_engine.train.__init__.**DummyTrain**

Bases: *Train*

This class is a decorator for the *Train* class that creates a "dummy" train. The *DummyTrain* class inherits from the *Train* class and overrides some of its methods to provide a simplified implementation for testing or other purposes.

The *log()* method is a no-op, and the *should_log()* method always returns *False*, indicating that this train should not be logged.

The *layover_and_turnback()* method calls the *delete()* method, which removes the train from the simulation.

layover_and_turnback() → None

log() → None

should_log() → bool

simulation: *Simulation*

class transit_lab_simmetro.simulation_engine.train.__init__.**DummyTrainDecorator**

Bases: *DummyTrain*

The *DummyTrainDecorator* class is a decorator for the *Train* class that creates a "dummy" train. It inherits from the *DummyTrain* class and overrides some of its methods to provide a simplified implementation for testing or other purposes.

The *__init__* method copies the properties from the *DummyTrain* instance to the *DummyTrainDecorator* instance, sets the *train_id* to a unique identifier, initializes the *state* to *WaitingToBeDispatched*, and registers the train with the *train_speed_regulator*.

The *simulation* attribute is a reference to the *Simulation* object that the *DummyTrain* instance is part of, allowing the *DummyTrain* to interact with the overall simulation, such as removing itself from the simulation when the *layover_and_turnback()* method is called.

__init__(*train:* Train)

This method initializes a new instance of the *DummyTrainDecorator* class. It copies the properties from the provided *Train* instance to the *DummyTrainDecorator* instance, sets a unique identifier for the *train_id*, initializes the *state* to *WaitingToBeDispatched*, and registers the train with the *train_speed_regulator*.

simulation: *Simulation*

class transit_lab_simmetro.simulation_engine.train.__init__.**Train**

Bases: `object`

Represents a train in the simulation. The *Train* class handles the train's movement, speed regulation, passenger management, and logging.

The *Train* class has the following key responsibilities: - Manages the train's state and transitions between different states (e.g. waiting to be dispatched, dwelling at a station). - Updates the train's speed and distance travelled based on the current block and speed regulator. - Tracks the train's location and progress along the path. - Interacts with the train's passenger manager to handle passenger boarding and alighting. - Logs the train's status and updates to the simulation logger.

__init__(*train_speed_regulator:* TrainSpeedRegulator | TrainSpeedRegulatorCTA, *train_passenger_manager:* TrainPassengerManager, *path:* Path, *starting_block_index: int = 0*, *dispatching_time: float | None = None*, *runid: str | None = None*)

Initializes a new instance of the *Train* class.

Parameters

- **train_speed_regulator** (TrainSpeedRegulator | TrainSpeedRegulatorCTA) – The speed regulator for the train.
- **train_passenger_manager** (TrainPassengerManager) – The passenger manager for the train.
- **path** (Path) – The path the train will follow.
- **starting_block_index** (*int, optional*) – The index of the starting block on the path. Defaults to 0.
- **dispatching_time** (*float, optional*) – The time when the train will be dispatched.
- **runid** (*str, optional*) – The unique identifier for the train.

train_id

The unique identifier for the train.

Type

str

`steps_since_last_log`

The number of steps since the last log update.

Type
int

`train_speed_regulator`

The speed regulator for the train.

Type
TrainSpeedRegulator | *TrainSpeedRegulatorCTA*

`passenger_manager`

The passenger manager for the train.

Type
TrainPassengerManager

`path`

The path the train will follow.

Type
Path

`has_been_short_turned`

Indicates whether the train has been short-turned.

Type
bool

`starting_block_index`

The index of the starting block on the path.

Type
int

`current_block_index`

The index of the current block on the path.

Type
int

`distance_travelled_in_current_block`

The distance the train has travelled in the current block.

Type
float

`speed`

The current speed of the train.

Type
float

`acceleration`

The current acceleration of the train.

Type
float

`dispatching_time`

The time when the train will be dispatched.

Type
float

state

The current state of the train.

Type
TrainState

length

The length of the train.

Type
float

_should_log

Indicates whether the train should be logged.

Type
bool

property acceleration_in_fps2: float

block_with_red_signals_in_sight() → Tuple[float | None, int | None]

property current_block: BlockType

property current_speed_code: float

delete() → None

property distance_to_next_block: float

property distance_to_next_station: float

distance_traveled_from_the_start_of_block(*asking_block: BlockType*) → float

Calculates the distance traveled by the train from the start of the specified block.

Parameters
asking_block (*BlockType*) – The block for which the distance traveled from the start is to be calculated.

Returns
The distance traveled by the train from the start of the specified block.

Return type
float

property first_block_after_station: BlockType | None

classmethod generate_train_id()

get_all_stops_ahead() → list[*Station*]

get_next_station() → *Station*

property location_from_terminal: float

log() → None

property next_block: BlockType

property previous_block: BlockType

set_state_to_dwelling_at_station(*station:* Station) → None

should_log() → bool

simulation: *Simulation*

property speed_in_fps: float

property time_step: float

property total_travelled_distance: float

property total_travelled_distance_from_dispatch: float

train_logger: TrainLogger | None = None

update() → None

update_block() → None

This method updates the train's position within the current block and moves the train to the next block if the distance traveled in the current block exceeds the block's length.

It first checks if the distance traveled in the current block is greater than or equal to the block's length. If so, it subtracts the block's length from the distance traveled in the current block and increments the current block index to move the train to the next block.

It then calculates the position of the train's rear relative to the start of the current block and iterates backwards through the blocks, deactivating the blocks that the train's rear has left behind.

update_distance_travelled() → float

update_speed() → None

class transit_lab_simmetro.simulation_engine.train.__init__.**TrainPassengerManager**

Bases: object

__init__(*train_capacity: int*, *num_cars: int = 8*, *num_doors_per_car: int = 2*, *car_capacity: int = None*, *num_seats_per_door: int = 20*)

alight_all_passengers(*current_station*, *current_time*)

alight_passengers(*current_station*, *current_time*)

board_passengers(*passengers*, *current_time*, *car_assignment_weights: List[float] = [1, 3, 1, 1, 1, 1, 3, 1]*)

get_door_metrics(*alight_counts*, *boarding_counts*)

remaining_capacity() → int

property total_passengers

class transit_lab_simmetro.simulation_engine.train.__init__.**TrainSpeedRegulator**

Bases: object

__init__(*max_acceleration: float*, *normal_deceleration: float*, *emergency_deceleration: float*)

property braking_distance: float

entered_symptomatic_block(*symptomatic_block*) → None

property normal_acceleration: float

property normal_acceleration_in_fps2

property normal_decceleration_in_fps2

property planning_distance: float

register_train(*train:* Train) → None

regulate_acceleration()

property train: *Train*

train_stopped_at_station(*station:* Station) → None

update_train_speed() → None

class transit_lab_simmetro.simulation_engine.train.__init__.TrainSpeedRegulatorCTA

Bases: object

__init__(*max_acceleration: float*, *normal_deceleration: float*, *emergency_deceleration: float*, *desired_speed_range: tuple[float, float] = (0.8, 1.0)*)

property braking_distance: float

property braking_distance_for_station: float

entered_symptomatic_block(*symptomatic_block:* OffScanSymptomaticBlockDecorator) → None

property normal_acceleration: float

property normal_acceleration_in_fps2

property normal_decceleration_in_fps2

property planning_distance: float

register_train(*train:* Train) → None

regulate_acceleration()

property train: *Train*

train_stopped_at_station(*station:* Station) → None

update_desired_speed() → None

update_train_speed() → None

6.2 Data Flow

Explain how data flows through the system.

CHAPTER

SEVEN

DEVELOPMENT

7.1 Development Environment Setup

Guide on setting up the development environment.

7.2 Build Instructions

Explain how to build the project from source.

7.3 Testing

Describe how to run tests.

CHAPTER

EIGHT

CONTRIBUTION GUIDELINES

8.1 How to Contribute

Instructions for making contributions, including coding standards and the pull request process.

8.2 Community Guidelines

Code of conduct and how to get involved in the community.

CHAPTER

NINE

FAQS

Address common questions and issues.

CHAPTER

TEN

TROUBLESHOOTING

10.1 Common Issues

List common issues and their solutions.

10.2 Getting Help

Information on where to ask questions or report issues.

CHAPTER

ELEVEN

CHANGELOG

11.1 Version History

List of changes for each version, including new features, bug fixes, and breaking changes.

CHAPTER

TWELVE

APPENDIX

12.1 Glossary

Definitions of terms used in the documentation.

12.2 Further Reading

Links to additional resources such as blog posts, tutorials, and papers.

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[KW07] Haris N. Koutsopoulos and Zhigao Wang. Simulation of Urban Rail Operations. *Transportation Research Record*, 2006:84–91, 2007. URL: https://api.semanticscholar.org/CorpusID:110620690.

[WK11] Zhigao Wang and Haris N. Koutsopoulos. Calibration of urban rail simulation models: a methodology using SPSA algorithm. In *Proceedings of the Winter Simulation Conference*, Wsc '11, 3704–3714. Winter Simulation Conference, 2011.

[Zho22] Jiali Zhou. *Urban rail simulation and applications in service planning and operations*. PhD thesis, Northeastern University, 2022.

[ZK22] Jiali Zhou and Haris N. Koutsopoulos. Schedule-based Analysis of Transmission Risk in Public Transportation Systems. *ArXiv*, 2022. URL: https://api.semanticscholar.org/CorpusID:246904579.

[ZKS20] Jiali Zhou, Haris N. Koutsopoulos, and Saeid Saidi. Evaluation of Subway Bottleneck Mitigation Strategies using Microscopic, Agent-Based Simulation. *Transportation Research Record*, 2674:649–661, 2020. URL: https://api.semanticscholar.org/CorpusID:218922083.

PYTHON MODULE INDEX

t

INDEX

Symbols

A

J

L

M

N

T

U

W