Resources

Python 3 Google Compute Engine backend
Showing resources since 7:15 PM

RAM

Disk

Libraries

```
!pip install simpy
#Importing all the libraries required for the project.
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.stats as stats
import math
import numpy as np
import random

import simpy
```

```
Collecting simpy
  Downloading https://files.pythonhosted.or
Installing collected packages: simpy
Successfully installed simpy-4.0.1
```

```
ε = 0.00001

def isZero(x):
    return abs(x)<ε
```

Entities

```
# Time tolerance: when at current speed difference a crash might occur within that number of
CRITICAL_TIME_TOLERANCE = 4  # [s]
LANE_CHANGE_TIME = 2 # [s]
MIN_TIME_DIFF = 0.5

MIN_SPEED_DIFF = 3 # [m/s] min speed diff to trigger overtaking

CAR_LENGTH = 4 # [m]

FAR_AWAY_IN_FRONT = 200 # [m]  distance at which a car in front can be ignored
FAR_AWAY_IN_BACK = 80   # [m]  distance at which a car behind can be ignored
```

Lanes

Vehicles

```
def isRunning(p):
    return p is not None and p.running

def isCrashed(p):
    return p is not None and p.crashed


VEHICLE_ID = 0

class Vehicle:
    def __init__(self, env, rec, types,
                 startingLane=None, startingPos=0,
                 t0=0, x0=0, dx0=0, ddx0=0, dddx0=0,
                 t=[], v=[]):

        global VEHICLE_ID
        self.id = VEHICLE_ID
        VEHICLE_ID += 1

        if types=="passenger":
            self.a_min = -4.3 # [m/s²]
            self.a_max = 3 # [m/s²] corresponds to 0-100km/h om 12s

        if types == "commercialV":
            self.a_min = -0.85 # [m/s²]
            self.a_max = 0.75 # [m/s²] corresponds to 0-100km/h om 12s

        if types == "sports":
            self.a_min = -8 # [m/s²]
            self.a_max = 7.5 # [m/s²] corresponds to 0-100km/h om 12s

        self.env = env
        self.rec = rec

        self.startingLane = startingLane
        self.startingPos = startingPos
        self.lane = None
        self.pos = 0

        ## second lane reference during changing of lanes
        self.oldLane = None

        self.t0 = t0
        self.x0 = x0
```

```
        self.x0 = x0
        self.dx0 = dx0
        self.ddx0 = ddx0
        self.dddx0 = dddx0

        self.t = t
        self.v = v
        self.t_target = []
        self.v_target = []

        self.running = False
        self.crashed = False
        self.braking = False
        self.changingLane = False

        self.processRef = None
        self.env.process(self.process())

        ## this allows to trigger trace messages for
        ## the new feature Surround
        self.traceSurround = False
        self.traceOvertake = False
        self.traceBrake = False

    def __str__(self):
        return f"({self.id:d})"

    def isNotFasterThan(self, other):
        return True if other is None else self.dx0 <= other.dx0

    def isNotSlowerThan(self, other):
        return True if other is None else other.dx0 <= self.dx0

    def updateOnly(self):
        if self.crashed:
            return False
        t = self.env.now
        if t < self.t0:
            return False
        if self.running and t > self.t0:
            dt = t - self.t0
            ddx = self.ddx0 + self.dddx0*dt
            dx = round(self.dx0 + self.ddx0*dt + self.dddx0*dt*dt/2,4)
            Δx = self.dx0*dt + self.ddx0*dt*dt/2 + self.dddx0*dt*dt*dt/6
            x = round(self.x0 + Δx, 2)
            self.t0, self.x0, self.dx0, self.ddx0 = t, x, dx, ddx

            self.pos = round(self.pos+Δx, 2)
            # update lane information if necessary
            if self.pos >= self.lane.length:
                nextPos = self.pos - self.lane.length
                nextLane = self.lane.next
```

```
                self.lane.leave(self)
                if nextLane is None:
                    self.rec.record(self, event='end')
                    self.running = False
                    return False
                else:
                    nextLane.enter(self, pos=nextPos)
        return True

    def update(self):
        active = self.updateOnly()
        if not active:
            return False

        self.surround = Surround(self)

        ## instead of direct link, call method
        inFront = self.surround.front
        if (isRunning(inFront) or isCrashed(inFront)) \
               and inFront.x0 < self.x0 + CAR_LENGTH:
            self.crash(inFront)
            return True

        if inFront is not None and not self.braking and \
                self.dx0 > inFront.dx0 and \
                self.x0 + CRITICAL_TIME_TOLERANCE*self.dx0 > inFront.x0:
            Δt = max(MIN_TIME_DIFF, (inFront.x0-self.x0)/self.dx0)
            self.setTarget(Δt, inFront.dx0)
            self.interruptProcess()
            return True

        ## new code: start overtaking maneuver by changing into fast lane
        if inFront is not None and \
                not self.braking and not self.changingLane and \
                self.dx0 > inFront.dx0 + MIN_SPEED_DIFF and \
                self.x0 + (LANE_CHANGE_TIME+CRITICAL_TIME_TOLERANCE)*self.dx0 > inFront.x0 an
                self.surround.rightLane is not None and \
                self.surround.right is None and \
                self.isNotFasterThan(self.surround.rightFront) and \
                self.isNotSlowerThan(self.surround.rightBack):
            if self.traceOvertake:
                print(f"t={self.t0:7,.1f}s Overtaking v{self.id:d} overtakes v{inFront.id:d}
            self.setTarget(LANE_CHANGE_TIME, 'fast')
            self.interruptProcess()
            return True

        ## new code: end overtaking by returning to slow lane
        if self.surround.leftLane is not None and \
                not self.braking and not self.changingLane and \
                self.surround.left is None and \
                self.isNotFasterThan(self.surround.leftFront) and \
                self.surround.leftBack is None:
```

```
            self.surround.leftback is None:
        if self.traceOvertake:
            print(f"t={self.t0:7,.1f}s Overtaking v{self.id:d} returns to slow lane at x=
        self.setTarget(LANE_CHANGE_TIME, 'slow')
        self.interruptProcess()
        return True

    def setTarget(self, Δt, v):
        self.t_target = [ Δt ] + self.t_target
        self.v_target = [ v ] + self.v_target

    def process(self):

        # delay start to the given time t-
        if self.t0>self.env.now:
            yield self.env.timeout(self.t0-self.env.now)
        self.t0 = self.env.now
        self.running = True
        self.rec.startRecording(self)
        self.startingLane.enter(self, pos=self.startingPos)

        while self.running:
            self.updateOnly()

            self.surround = Surround(self)

            inFront = self.surround.front
            if inFront is not None:

                # if the car in front is slower and we are a bit too near on its heals...
                if inFront.dx0 < self.dx0 and \
                        inFront.x0 < self.x0 + CRITICAL_TIME_TOLERANCE*self.dx0:
                    if self.traceBrake:
                        print(f"t={self.t0:7,.1f}s Braking v{self.id:d} v={self.dx0:4.4f}m/s


                    yield from self.emergencyBraking(inFront.dx0)
                    if not isZero(self.dx0-inFront.dx0):
                        # after emergency breaking adjust to the speed of the car in front...
                        Δt = 2
                        self.setTarget(Δt, inFront.dx0)
                    continue

            if len(self.t_target)==0:
                self.t_target = self.t.copy()
                self.v_target = self.v.copy()

            if len(self.t_target)>0:

                ## add code for explicit change of lane
                if type(self.v_target[0]) is str:
                    direction = normaliseDirection(self.v_target[0])
```

```
                    t = self.t_target[0]
                    self.t_target = self.t_target[1:]
                    self.v_target = self.v_target[1:]
                    if self.lane.getLane(direction) is not None:
                        yield from self.changeLane(direction, t)

                ## the rest is what was there before
                else:
                    v0 = self.dx0
                    v1 = self.v_target[0]
                    t = self.t_target[0]
                    self.t_target = self.t_target[1:]
                    self.v_target = self.v_target[1:]
                    if isZero(v1-v0):
                        yield from self.wait(t)
                    else:
                        yield from self.adjustVelocity(v1-v0, t)
            else:
                yield from self.wait(10)

        self.rec.stopRecording(self)

    def emergencyBraking(self, v):

        def emergencyBrakingProcess(v):
            self.rec.record(self, 'brake')
            minΔt = 0.2
            self.dddx0 = (self.a_min-self.ddx0)/minΔt
            yield self.env.timeout(minΔt)

            self.updateOnly()
            self.dddx0=0
            self.ddx0=self.a_min
            v = min(v, self.dx0-2)
                # the brake time estimate is for perfect timing for
                # autonomous cars. For manual driving leave out the
                # -minΔt/2 or use a random element.
            Δt = max(0.5, (v-self.dx0)/self.ddx0 - minΔt/2)
            yield self.env.timeout(Δt)

            self.updateOnly()
            self.dddx0 = -self.ddx0/minΔt
            yield self.env.timeout(minΔt)

            self.updateOnly()
            self.ddx0 = 0
            self.dddx0 = 0


        ## The 'braking' bit prevents the interruption of an emergency breaking process
        self.braking = True
        self.processRef = self.env.process(emergencyBrakingProcess(v))
```

```
        try:
            yield self.processRef
        except simpy.Interrupt:
            pass
        self.processRef = None
        self.braking = False

    ## make changeLane robust against interrupt:
    def changeLane(self, direction, Δt):

        # smoothly adjust velocity by Δv over the time Δt
        def changeLaneProcess(oldLane, newlane, Δt):
            self.updateOnly()
            self.rec.record(self, 'change '+direction)
            self.oldLane = oldLane
            newLane.enter(self, pos=self.pos)
            self.ddx0 = 1
            self.dddx0 = 0
            yield self.env.timeout(Δt)
            self.updateOnly()
            self.oldLane.leave(self)
            self.lane = newLane
            self.oldLane = None
            self.rec.record(self, 'done change '+direction)
            self.updateOnly()
            self.ddx0 = 0
            self.dddx0 = 0

        ## keep record of current lane, as in case of aborting
        ## the lane change
        ## when interrupted go back into original lane
        oldLane = self.lane
        newLane = self.lane.getLane(direction)
        self.changingLane = True
        try:
            self.processRef = self.env.process(changeLaneProcess(oldLane, newLane, Δt))
            yield self.processRef
            self.processRef = None
        except simpy.Interrupt:
            # if interrupted go quickly back into old lane
            # but this is not interruptible
            self.processRef = None
            self.env.process(changeLaneProcess(newLane, oldLane, Δt/4))
        self.changingLane = False

    def adjustVelocity(self, Δv, Δt):

        # smoothly adjust velocity by Δv over the time Δt
        def adjustVelocityProcess():
            self.updateOnly()
            minΔt = 0.1*Δt
```

```
            a = Δv/(Δt-minΔt)
            tt = Δt-2*minΔt

            self.dddx0 = (a-self.ddx0)/minΔt
            yield self.env.timeout(minΔt)

            self.updateOnly()
            self.dddx0 = 0
            self.ddx0 = a
            yield self.env.timeout(tt)

            self.updateOnly()
            self.dddx0 = -a/minΔt
            yield self.env.timeout(minΔt)

            self.updateOnly()
            self.dddx0 = 0
            self.ddx0 = 0

        self.processRef = self.env.process(adjustVelocityProcess())
        try:
            yield self.processRef
        except simpy.Interrupt:
            self.dddx0 = 0
            pass
        self.processRef = None

    def wait(self, Δt):

        def waitProcess():
            yield self.env.timeout(Δt)

        self.processRef = self.env.process(waitProcess())
        try:
            yield self.processRef
        except simpy.Interrupt:
            pass
        self.processRef = None

    def interruptProcess(self):
        if self.processRef is not None and self.processRef.is_alive:
            self.processRef.interrupt('change')

    def crash(self, other):

        def recordCrash(self):
            self.rec.record(self, 'crash')
            self.running = False
            self.crashed = True
            self.dx0 = 0
            self.ddx0 = 0
            self.dddx0 = 0
```

```
        if self.running:
            print(f"Crash p{self.id:d} into p{other.id:d} at t={self.t0:7.3f} x={self.x0:7.1f
            recordCrash(self)
            if other.running:
                recordCrash(other)
```

Surroundings of car

```
class Surround:

    def __init__(self, vehicle):

        def s(vehicle):
            if vehicle is None:
                return " "
            elif type(vehicle) is list:
                if len(vehicle)==1:
                    return s(vehicle[0])
                else:
                    res = "["
                    for v in vehicle:
                        if len(res)>1:
                            res += ','
                        res+=s(v)
                    res += "]"
                    return res
            else:
                return f"{vehicle.id:d}"

        # For each of the directions None means that there is no
        # vehicle in the immediate vicinity.
        # We initialise to a 'safe' value which can be easily detected
        # if something goes wrong

        self.leftBack = vehicle
        self.left = vehicle
        self.leftFront = vehicle
        self.back = vehicle
        self.vehicle = vehicle
        self.front = vehicle
        self.rightBack = vehicle
        self.right = vehicle
        self.rightFront = vehicle

        lane = vehicle.lane
        pos = vehicle.pos
        if lane is not None:
            self.lane = lane
```

```
            self.front = lane.inFront(pos)
            self.back = lane.behind(pos)

            self.rightLane = lane.right
            if self.rightLane is not None:
                if vehicle.oldLane == lane.right:
                    # drifting left
                    self.right = vehicle
                    self.rightFront = self.rightLane.inFront(pos)
                    self.rightBack = self.rightLane.behind(pos)
                else:
                    right = self.rightLane.at(pos)
                    if len(right)==0:
                        self.right = None
                    elif len(right)==1:
                        self.right = right[0]
                    else:
                        self.right = right

                    if self.right is None:
                        self.rightFront = self.rightLane.inFront(pos)
                        self.rightBack = self.rightLane.behind(pos)
                    else:
                        self.rightFront = None
                        self.rightBack = None

            self.leftLane = lane.left
            if self.leftLane is not None:
                if vehicle.oldLane == lane.left:
                    # drifting right
                    self.left = vehicle
                    self.leftFront = self.leftLane.inFront(pos)
                    self.leftBack = self.leftLane.behind(pos)
                else:
                    left = self.leftLane.at(pos)
                    if len(left)==0:
                        self.left = None
                    elif len(left)==1:
                        self.left = left[0]
                    else:
                        self.left = left

                    if self.left is None:
                        self.leftFront = self.leftLane.inFront(pos)
                        self.leftBack = self.leftLane.behind(pos)
                    else:
                        self.leftFront = None
                        self.lefttBack = None

        if vehicle.traceSurround:
            print(f"surround t={self.vehicle.env.now:6.2f} " +
                  "|" +
```

```
                 |  +
                ("" if self.leftLane is None else
                    f"|{s(self.leftBack):s}>{s(self.left):s}>{s(self.leftFront):s}") +
                f"|{s(self.back):s}>{s(self.vehicle):s}>{s(self.front):s}|" +
                ("" if self.rightLane is None else
                    f"{s(self.rightBack):s}>{s(self.right):s}>{s(self.rightFront):s}|") +
                "|"
               )
```

Recorder

[ ] ↳ *1 cell hidden*

Different Vehicles with Fixed Speed

```
l = Lane(1000, VMAX)
while l.totalLength()<3000:
    l.extend(Lane(1000, VMAX))
r = l.widenRight()


VMAX = 120/3.6
N = 55
DT = 14 # time difference between start
env = simpy.Environment()
rec = SimpleRecorder(env, 0, 800, 1)

for i in range(N):
    n = random.randint(1,3)
    if n==1:
      CAR_LENGTH = 4.5
      LANE_CHANGE_TIME = 2
      FAR_AWAY_IN_FRONT = 200 # [m]  distance at which a car in front can be ignored
      FAR_AWAY_IN_BACK = 80   # [m]  distance at which a car behind can be ignored
      v = Vehicle(env, rec, "passenger",startingLane=l, t0=i*DT, dx0=VMAX, t=[10], v=[20])
    elif n==2:
      CAR_LENGTH = 10
      LANE_CHANGE_TIME = 5
      FAR_AWAY_IN_FRONT = 100 # [m]  distance at which a car in front can be ignored
      FAR_AWAY_IN_BACK = 40   # [m]  distance at which a car behind can be ignored
      v = Vehicle(env, rec, "commercialV",startingLane=l, t0=i*DT, dx0=VMAX, t=[30], v=[14])
    else:
      LANE_CHANGE_TIME = 1.5
      CAR_LENGTH = 5
```

```
        FAR_AWAY_IN_FRONT = 400 # [m]  distance at which a car in front can be ignored
        FAR_AWAY_IN_BACK = 160   # [m]  distance at which a car behind can be ignored
        v = Vehicle(env, rec, "sports",startingLane=l, t0=i*DT, dx0=VMAX, t= [5], v=[30])

    v.traceOvertake = True
rec.run()
```

```
t=  586.0s Overtaking v39 returns to slow
t=  593.0s Overtaking v41 returns to slow
t=  594.0s Overtaking v39 overtakes v35 a
t=  602.0s Overtaking v41 overtakes v40 a
t=  609.0s Overtaking v39 returns to slow
t=  614.0s Overtaking v36 overtakes v31 a
t=  619.0s Overtaking v43 overtakes v12 a
t=  626.0s Overtaking v41 returns to slow
t=  626.0s Overtaking v43 returns to slow
t=  630.0s Overtaking v43 overtakes v42 a
t=  633.0s Overtaking v44 overtakes v12 a
t=  640.0s Overtaking v44 returns to slow
t=  643.0s Overtaking v39 overtakes v31 a
t=  645.0s Overtaking v36 returns to slow
t=  645.0s Overtaking v44 overtakes v42 a
t=  648.0s Overtaking v43 returns to slow
t=  650.0s Overtaking v41 overtakes v38 a
t=  653.0s Overtaking v43 overtakes v40 a
t=  654.0s Overtaking v46 overtakes v12 a
t=  656.0s Overtaking v44 returns to slow
t=  663.0s Overtaking v46 returns to slow
t=  664.0s Overtaking v44 overtakes v40 a
t=  666.0s Overtaking v43 returns to slow
t=  668.0s Overtaking v47 overtakes v12 a
t=  672.0s Overtaking v46 overtakes v45 a
t=  675.0s Overtaking v43 overtakes v38 a
t=  677.0s Overtaking v47 returns to slow
t=  679.0s Overtaking v44 returns to slow
t=  681.0s Overtaking v46 returns to slow
t=  687.0s Overtaking v47 overtakes v45 a
t=  696.0s Overtaking v49 overtakes v12 a
t=  704.0s Overtaking v47 returns to slow
t=  705.0s Overtaking v49 returns to slow
t=  713.0s Overtaking v41 returns to slow
t=  714.0s Overtaking v49 overtakes v48 a
t=  717.0s Overtaking v50 overtakes v12 a
t=  722.0s Overtaking v46 overtakes v42 a
t=  723.0s Overtaking v49 returns to slow
t=  724.0s Overtaking v50 returns to slow
t=  724.0s Overtaking v51 overtakes v12 a
t=  729.0s Overtaking v50 overtakes v48 a
t=  733.0s Overtaking v51 returns to slow
t=  740.0s Overtaking v50 returns to slow
t=  748.0s Overtaking v50 overtakes v49 a
t=  752.0s Overtaking v46 returns to slow
t=  764.0s Overtaking v49 overtakes v45 a
t=  765.0s Overtaking v51 overtakes v48 a
t=  769.0s Overtaking v47 overtakes v42 a
t=  773.0s Overtaking v54 overtakes v12 a
t=  776.0s Overtaking v50 returns to slow
```

```
t=  780.0s Overtaking v54 returns to slow
t=  784.0s Overtaking v54 overtakes v53 a
t=  793.0s Overtaking v51 returns to slow
t=  794.0s Overtaking v49 returns to slow
t=  796.0s Overtaking v54 returns to slow
t=  797.0s Overtaking v50 overtakes v42 a
t=  799.0s Overtaking v47 returns to slow
```

```
df = rec.getData()
```

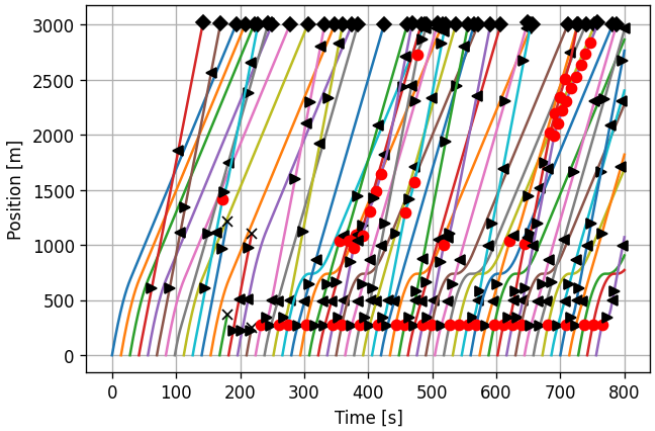
```
start, end = {},{}
time_taken = []
for i in range(len(df)):
    if df["id"][i] not in start:
        start[df["id"][i]] = df["t"][i]
        end[df["id"][i]] = df["t"][i]
    end[df["id"][i]] = df["t"][i]

for i in start:
    time_taken.append(end[i]-start[i])
#print(start)
#print(end)
average = sum(time_taken) / len(time_taken)
print(average)
```

```
141.52487640335025
```

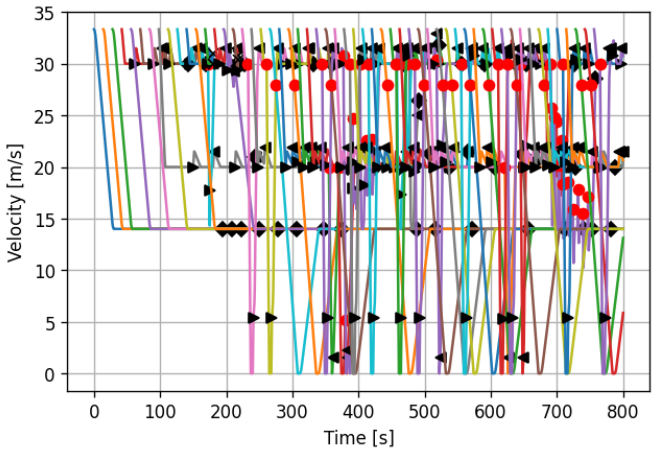
```
event = "end"
for index, row in df.iloc[::-1].iterrows():
    if df["event"][index]=="end":
        number = df["id"][index]
        time = df["t"][index]
        break

#print(number)
#print(time)
factor = 3600/time
#print(factor)
throughput = number*factor
print(f"The Throughput is {throughput} cars per hour")
```

```
The Throughput is 210.1522842639594 cars pe
```

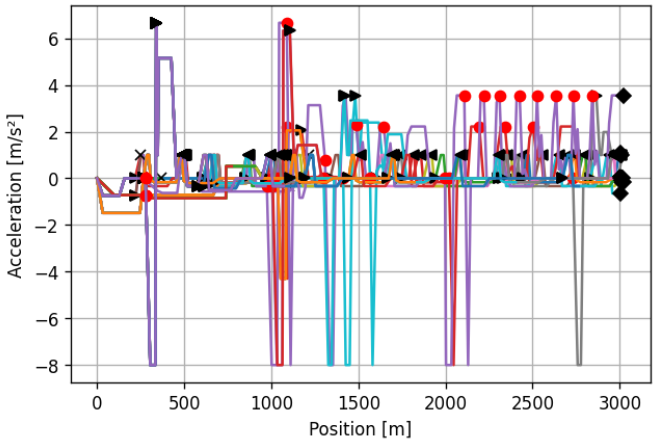
```
rec.plot('t', 'x')
```



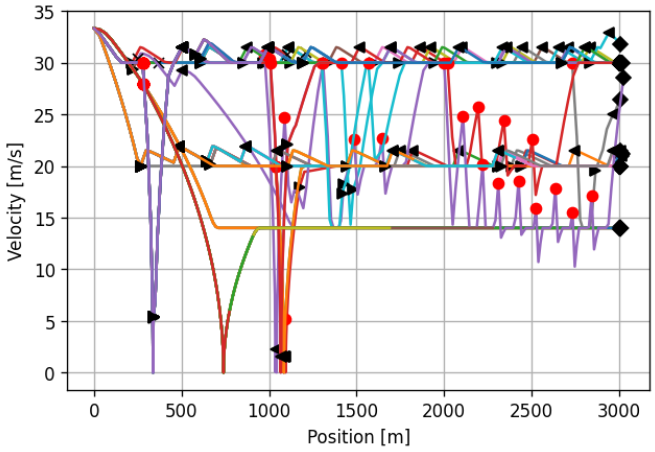

```
rec.plot('t', 'v')
```



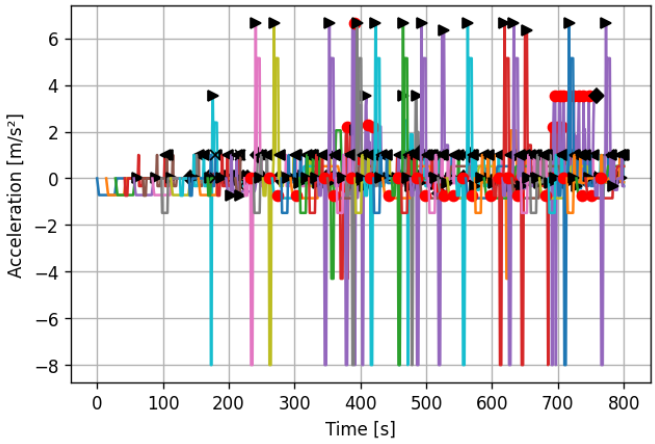

```
rec.plot('x', 'a')
```


Acceleration [m/s²]
Position [m]
6
4
2
0
−2
−4
−6
−8
0
500
1000
1500
2000
2500
3000

```
rec.plot('x', 'v')
```


Velocity [m/s]
Position [m]
35
30
25
20
15
10
5
0
0
500
1000
1500
2000
2500
3000

```
rec.plot('t', 'a')
```




```
SLOW_CYCLE = 100


def randomIntervals(cycles):
    # return [ random.expovariate(1.0/SLOW_CYCLE)+10 for i in range(cycles) ]
    return [ max(0, random.normalvariate(SLOW_CYCLE, SLOW_CYCLE/3)) for i in range(cycles) ]


SPEED_VARIATION = 0.05
def randomSpeedVariation(vmax, cycles, cv=SPEED_VARIATION):
    return [ vmax + (-1)**i*abs(random.normalvariate(0, vmax*cv)) for i in range(cycles) ]


#Creating Lanes
```

```
VMAX= 120/3.6
l = Lane(3000, VMAX)
r = l.widenRight()
print("Left Lane: ", l)
print("Right Lane:", r)
```

```
     Left Lane:  [0 3000m R:1]
     Right Lane: [1 3000m L:0]
```

```
VMAX = 120/3.6
N = 55 # number of points
DT =  15# time difference between start
random.seed(13)
env = simpy.Environment()
rec = SimpleRecorder(env, 0, 1000, 1)

for i in range(N):
    CYCLES = random.randint(1, 3)
    times = randomIntervals(CYCLES)
    speed = randomSpeedVariation(VMAX, CYCLES)
    n = random.randint(1,3)
    if n==1:
      CAR_LENGTH = 4.5
      LANE_CHANGE_TIME = 2
      FAR_AWAY_IN_FRONT = 200 # [m]  distance at which a car in front can be ignored
      FAR_AWAY_IN_BACK = 80   # [m]  distance at which a car behind can be ignored
      v = Vehicle(env, rec, "passenger",startingLane=l, t0=i*DT, dx0=speed[-1], t=min(times,[
    elif n==2:
      CAR_LENGTH = 10
      LANE_CHANGE_TIME = 5
      FAR_AWAY_IN_FRONT = 100 # [m]  distance at which a car in front can be ignored
      FAR_AWAY_IN_BACK = 40   # [m]  distance at which a car behind can be ignored
      v = Vehicle(env, rec, "commercialV",startingLane=l, t0=i*DT, dx0=VMAX, t=min(times,[30]
    else:
      LANE_CHANGE_TIME = 1.5
      CAR_LENGTH = 5
      FAR_AWAY_IN_FRONT = 400 # [m]  distance at which a car in front can be ignored
      FAR_AWAY_IN_BACK = 160   # [m]  distance at which a car behind can be ignored
      v = Vehicle(env, rec, "sports",startingLane=l, t0=i*DT, dx0=VMAX, t=min(times,[5]), v=m

    v.traceOvertake = True
rec.run()
```

```
     t=  102.0s Overtaking v4 overtakes v1 at
     t=  130.0s Overtaking v4 returns to slow
     t=  132.0s Overtaking v6 overtakes v5 at
     t=  169.0s Overtaking v6 returns to slow
     t=  190.0s Overtaking v6 overtakes v3 at
     t=  193.0s Overtaking v10 overtakes v9 at
     t=  227.0s Overtaking v6 returns to slow
     t=  229.0s Overtaking v14 overtakes v13 a
     t=  257.0s Overtaking v15 overtakes v13 a
     t=  271.0s Overtaking v14 returns to slow
```

```
t=  278.0s Overtaking v14 overtakes v9 at
t=  285.0s Overtaking v16 overtakes v13 a
t=  299.0s Overtaking v15 returns to slow
t=  306.0s Overtaking v14 returns to slow
t=  306.0s Overtaking v15 overtakes v9 at
t=  313.0s Overtaking v17 overtakes v13 a
t=  315.0s Overtaking v20 overtakes v19 a
t=  321.0s Overtaking v15 returns to slow
t=  327.0s Overtaking v16 returns to slow
t=  340.0s Overtaking v20 returns to slow
t=  346.0s Overtaking v20 overtakes v18 a
t=  366.0s Overtaking v20 returns to slow
t=  417.0s Overtaking v25 overtakes v24 a
t=  468.0s Overtaking v26 overtakes v24 a
t=  469.0s Overtaking v30 overtakes v29 a
t=  497.0s Overtaking v31 overtakes v29 a
t=  498.0s Overtaking v30 returns to slow
t=  516.0s Overtaking v27 overtakes v24 a
t=  526.0s Overtaking v31 returns to slow
t=  544.0s Overtaking v35 overtakes v34 a
t=  555.0s Overtaking v31 overtakes v30 a
t=  561.0s Overtaking v31 returns to slow
t=  574.0s Overtaking v37 overtakes v36 a
t=  586.0s Overtaking v35 returns to slow
t=  590.0s Overtaking v37 returns to slow
t=  597.0s Overtaking v37 overtakes v34 a
t=  606.0s Overtaking v35 overtakes v29 a
t=  618.0s Overtaking v40 overtakes v39 a
t=  622.0s Overtaking v35 returns to slow
t=  638.0s Overtaking v37 returns to slow
t=  643.0s Overtaking v40 returns to slow
t=  647.0s Overtaking v38 overtakes v36 a
t=  662.0s Overtaking v43 overtakes v42 a
t=  663.0s Overtaking v41 overtakes v39 a
t=  672.0s Overtaking v40 overtakes v36 a
t=  685.0s Overtaking v38 returns to slow
t=  687.0s Overtaking v43 returns to slow
t=  695.0s Overtaking v39 overtakes v36 a
t=  707.0s Overtaking v44 overtakes v42 a
t=  713.0s Overtaking v41 returns to slow
t=  732.0s Overtaking v44 returns to slow
t=  752.0s Overtaking v45 overtakes v42 a
t=  754.0s Overtaking v49 overtakes v48 a
t=  796.0s Overtaking v49 returns to slow
t=  796.0s Overtaking v52 overtakes v51 a
t=  814.0s Overtaking v51 overtakes v50 a
t=  828.0s Overtaking v52 returns to slow
t=  835.0s Overtaking v52 overtakes v48 a
t=  845.0s Overtaking v51 returns to slow
```

```
df1=rec.getData()


start, end = {},{}
time_taken = []
for i in range(len(df1)):
```

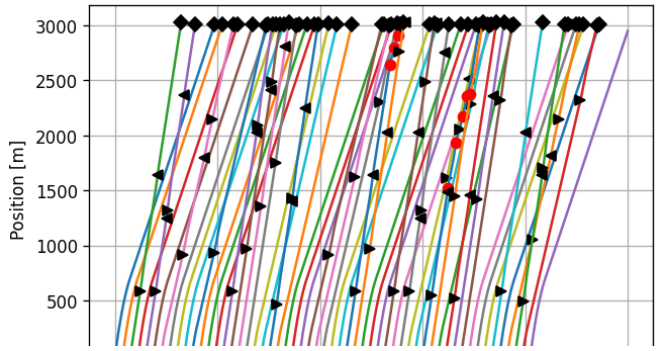
```
        if df1["id"][i] not in start:
            start[df1["id"][i]] = df1["t"][i]
            end[df1["id"][i]] = df1["t"][i]
        end[df1["id"][i]] = df1["t"][i]

    for i in start:
        time_taken.append(end[i]-start[i])
    print(start)
    print(end)
    average = sum(time_taken) / len(time_taken)
    print(average)
```

```
{0: 0, 1: 15, 2: 30, 3: 45, 4: 60, 5: 75, 6
{0: 194, 1: 209, 2: 129, 3: 239, 4: 156, 5:
151.76363636363635
```

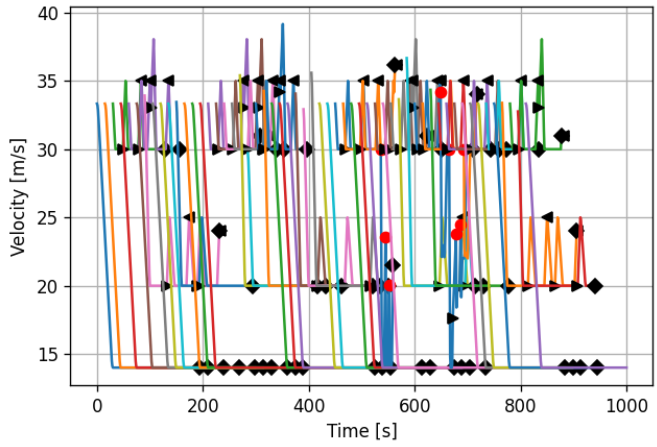
```
    event = "end"
    for index, row in df1.iloc[::-1].iterrows():
        if df1["event"][index]=="end":
            number = df1["id"][index]
            time = df1["t"][index]
            break

    #print(number)
    #print(time)
    factor = 3600/time
    #print(factor)
    throughput = number*factor
    print(f"The Throughput is {throughput} cars per hour")
```

```
The Throughput is 190.67796610169492 cars p
```

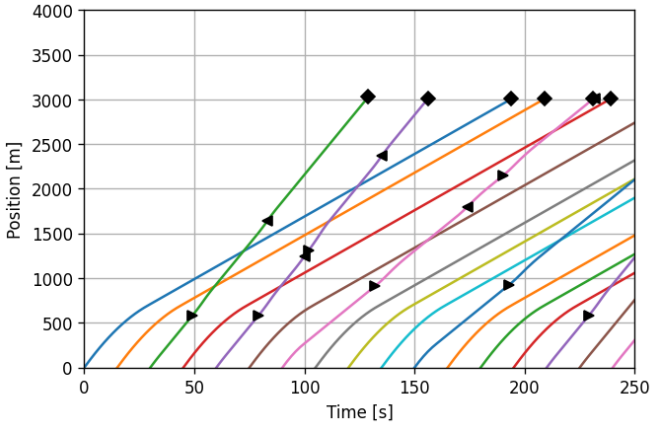
```
    rec.plot('t', 'x')
```



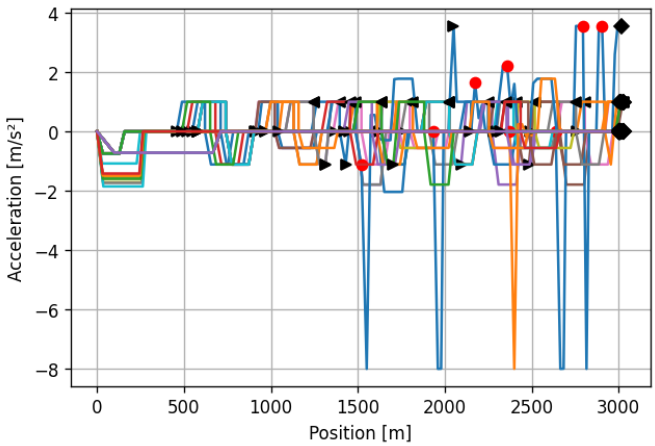

```
rec.plot('t', 'v')
```



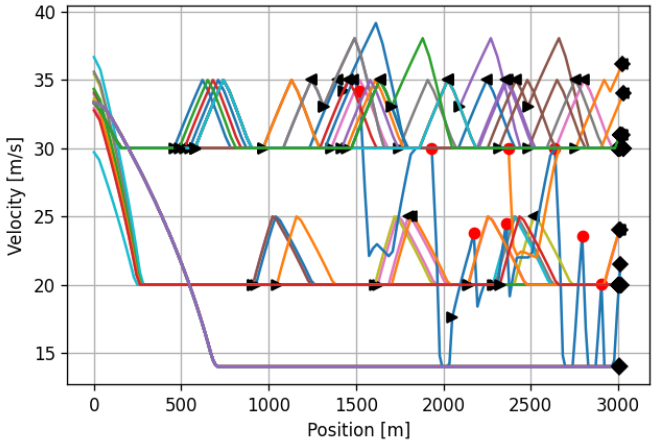

```
rec.plot('t', 'x', xmin=0, xmax=250, ymin=0, ymax=4000)
```




```
rec.plot('x', 'a')
```




```
rec.plot('x', 'v')
```



Manage Sessions Change Runtime Type

✓ 0s completed at 10:42 PM