

Rivaldo Lumelino, Alexandr Voronovich

11/23/2025

Professor Bandyopadhyay

Final Project

Interpretable Multi-Agent Coordination Algorithms for Heterogeneous (Urban/Suburban/Rural) Autonomous Mobility Networks.

The simulator behaves like a real V2X/MQTT-system.

Agent Class:

```
class Agent:
    def __init__(self, agent_id, x, y):
        self.id = agent_id # save agent id
        self.x = x # x coordinate
        self.y = y # y coordinate
        self.vx = 0 # x velocity
        self.vy = 0 # y velocity
        self.known_positions = {} # dictionary to store the last known positions from messages
        self.sensor_range = 5.0
        self.received_current_step = [] # buffer for messages received during current tick

        self.logs = [] # store human-readable reasoning messages
        self.prev_vx = 0 # previous x velocity
        self.prev_vy = 0 # previous y velocity

    def get_position(self): # helper: return current position as tuple
        return (self.x, self.y) # return x and y

    def update_velocity(self): # choose a new velocity by random position
        # store previous velocity so we can detect turns later
        self.prev_vx = self.vx
        self.prev_vy = self.vy
        self.vx = random.uniform(-1, 1) # set vx to random float between -1 to 1
        self.vy = random.uniform(-1, 1) # set vy to random float between -1 to 1

    def move(self): # update the position according to velocity
        self.x += self.vx # increment x by vx
        self.y += self.vy # increment y by vy

    def sense_agents(self, agents): # This function lets an agent detect other agents near them even if they have not been sensed yet
        sensed = [] # this is just an empty list, where we will store all nearby agents
        for agent in agents: # iterate through all agents passed in
            if agent.id == self.id: # skip sensing ourselves
                continue # continue to next agent
            distance = ((agent.x - self.x)**2 + (agent.y - self.y)**2)**0.5
            if distance <= self.sensor_range:
                sensed.append(agent)
        return sensed
```

Network Class

```
> v
class Network:
    def __init__(self, network_type):
        self.network_type = network_type # save what network is using urban, suburban, rural
        # Set network behavior based on type
        if network_type == "urban":
            self.drop_prob = 0.05 # 5% messages lost
            self.delay_range = (1, 3) # messages arrive in 1-3 steps (small delay)

        elif network_type == "suburban":
            self.drop_prob = 0.10 # 10% messages are lost
            self.delay_range = (2, 6) # messages arrive in 2-6 steps (medium delay)

        elif network_type == "rural":
            self.drop_prob = 0.30 # 30% message loss
            self.delay_range = (5, 15) # messages arrive in 5-15 steps (long delay)

        else:
            raise ValueError("Invalid Network Settings") # error if wrong network name is used

    # Message Queue
    self.queue = [] # list to store messages waiting to be delivered: tuples (deliver_time, age)
    self.time = 0 # keeps track of the current simulation time

    self.comm_success = 0 # how many messages were delivered successfully
    self.comm_attempts = 0 # how many messages were attempted to send
    self.collisions = 0 # how many collisions happened

    # This function simulates sending messages (now correctly inside the class)
    def broadcast(self, sender, msg, agents):
        for agent in agents: # loop through all agents
            if agent.id == sender.id: # skip sending a message to ourselves
                continue

            self.comm_attempts += 1 # count that a message was attempted

            if random.random() < self.drop_prob:
                continue

            if self.time < agent.receive_time:
                agent.receive_time = self.time + self.delay_range[1]
                agent.queue.append((self.time, msg))
                self.comm_success += 1
                self.collisions += 1
            else:
                agent.receive_time = self.time + self.delay_range[0]
                agent.queue.append((self.time, msg))
                self.comm_success += 1
                self.collisions += 1

    # This function checks if an agent has received a message
    def check_for_message(self, agent):
        if len(agent.queue) > 0:
            return True
        else:
            return False

    # This function removes a message from an agent's queue
    def remove_message(self, agent):
        if len(agent.queue) > 0:
            agent.queue.pop(0)
            return True
        else:
            return False

# This function checks if an agent has received a message
def check_for_message(self, agent):
    if len(agent.queue) > 0:
        return True
    else:
        return False

# This function removes a message from an agent's queue
def remove_message(self, agent):
    if len(agent.queue) > 0:
        agent.queue.pop(0)
        return True
    else:
        return False
```

Environment Class:

```
class Environment:
    # Create Agents and network
    def __init__(self, num_agents, network_type, steps=100):
        self.interpretability_log = {}
        self.agents = [
            Agent(
                agent_id=i, # give each agent ID
                x=random.uniform(0, 50), # random starting x position
                y=random.uniform(0, 50) # random y position
            )
            for i in range(num_agents) # repeat for all agents
        ]

        self.network = Network(network_type) # create a network with chosen type
        self.steps = steps # how many time steps to simulate

    # Collision Checker must be a method of Environment
    def check_collisions(self):
        positions = {} # store positions we have seen
        for agent in self.agents:
            pos = (round(agent.x, 1), round(agent.y, 1)) # round position to 1 decimal

            if pos in positions: # if another agent already has this spot
                self.network.collisions += 1 # count a collision
            else:
                positions[pos] = agent.id # store this position

    # main simulation loop as a method of Environment
    def run(self):
        for step in range(1, self.steps + 1):
            # Agents pick a new velocity
            for agent in self.agents:
                agent.update_velocity()

            # Agents send messages
            for agent in self.agents:
                msg = agent.create_message()

    ✓ 0.0s
```

Py

Output: Urban

```
[step 10] time=10 queue=110 comm_success=427/560 collisions=0
[step 20] time=20 queue=111 comm_success=960/1120 collisions=0
[step 30] time=30 queue=108 comm_success=1494/1680 collisions=0
[step 40] time=40 queue=111 comm_success=2032/2240 collisions=0
[step 50] time=50 queue=112 comm_success=2561/2800 collisions=0

===== SIMULATION SUMMARY =====
Network type: urban
Total steps: 50
Agents: 8
Messages sent: 2800
Messages recv'd: 2561
Collisions: 0
=====

==== SAMPLE INTERPRETABILITY LOGS (first 5 steps) ====

--- Time step 1, 8 events ---
[Agent 0] Slowed down due to no messages and no sensed agents. Velocity (-0.5591187559186066, 0.17853136775181744) -> (-0.11,0.04)
[Agent 1] Slowed down due to no messages and no sensed agents. Velocity (0.6188609133556533, -0.987002480643878) -> (0.12,-0.20)
[Agent 2] Slowed down due to no messages and no sensed agents. Velocity (0.6116385036656158, 0.3962787899764537) -> (0.12,0.08)
[Agent 3] Slowed down due to no messages and no sensed agents. Velocity (-0.31949896696401625, -0.6890410003764369) -> (-0.06,-0.14)
[Agent 4] Slowed down due to no messages and no sensed agents. Velocity (0.9144261444135624, -0.32681090977474647) -> (0.18,-0.07)

--- Time step 2, 21 events ---
[Agent 0] Received messages from agents [1, 5]
[Agent 0] Has comm info from agents [1, 5]. Using it for navigation.
[Agent 0] Large turn detected ( $\Delta v=1.03$ ). (-0.11,0.04) -> (-0.35,-0.96)
[Agent 1] Received messages from agents [3]
[Agent 1] Has comm info from agents [3]. Using it for navigation.
```

Explanation:

Urban Behavior messages delay 1-3 steps (Small queue → fast network)

[step 10]

- 10 steps out of 50

Time = 10

- Network time = 10 ticks

Queue = 110

- There are **110 messages still waiting** in the delivery queue
- These messages were sent but have not arrived yet

comm_success=427/560

- Out of **560** messages that agents sent so far:
 - **427** arrived successfully
 - **133** were dropped (because 5% drop rate)

collisions=0

- No two agents have hit the same position during the simulation

Network Type: Urban

Total Steps: 50

Agents: 8

Messages sent: 2800 (Each step, every agent sends to 7 others:

$8 \text{ agents} \times 7 \text{ receivers} \times 50 \text{ steps} = 2800$

Interpretability Logs — What Agents Were Thinking

Time step 1 — No one has any messages yet

Agents start with zero knowledge

[Agent 0] Slowed down due to no messages and no sensed agents.

Meaning:

- Agent 0 looked around → no one nearby
- No messages received yet (delay is 1–3 steps)
- To stay safe → slows down

Time step 2 — Some messages start arriving

[Agent 0] Received messages from agents [1, 5]

[Agent 0] Has comm info from agents [1, 5]. Using it for navigation.

Meaning:

- Communication started working
- Agent 0 now has info from agents 1 and 5
- Agent 0 will adjust movement based on that.

Large turn:

[Agent 0] Large turn detected ($\Delta v=1.03$). (-0.11,0.04) -> (-0.35,-0.96)

Meaning:

- The change in velocity was big
- Probably reacting to new info

Agent 1:

[Agent 1] Received messages from agents [3]

[Agent 1] Has comm info from agents [3]. Using it for navigation

Meaning:

- Agent 1 received only one message

Time step 3 - More messages, more clarity

[Agent 0] Received messages from agents [3, 3, 4, 5]

Meaning:

- Some duplicates appear due to multiple delays

[Agent 0] Has comm info from agents [1, 3, 4, 5]. Using it for navigation

Meaning:

- Agent 0 has a good mapping

[Agent 1] Received messages from agents [0, 2, 4, 4, 5]

[Agent 1] Has comm info from agents [0, 2, 3, 4, 5]. Using it for navigation.

[Agent 1] Large turn detected ($\Delta v=0.83$). (0.86,0.76) -> (0.31,0.13)

Meaning:

- Agent 1 good mapping
- Large turn detected(Reacting strongly to new information)

Time step 4 — Full communication

[Agent 0] Received messages from agents [1, 2, 2, 4, 6, 7]

[Agent 0] Has comm info from agents [1, 2, 3, 4, 5, 6, 7]. Using it for navigation.

[Agent 0] Large turn detected ($\Delta v=1.94$). (-0.92,-0.84) -> (-0.20,0.96)

Meaning:

- Agent 0 now knows where almost everyone is
- Very accurate motion
- Large turn detected(Reacting strongly to new information)

[Agent 1] Received messages from agents [0, 0, 2, 3, 5, 6, 6, 7, 7]

[Agent 1] Has comm info from agents [0, 2, 3, 4, 5, 6, 7]. Using it for navigation.

Meaning:

- Some duplicates appear due to multiple delays
- Agent 1 now knows where almost everyone is

Time step 5 — Continuous full communication

[Agent 0] Received messages from agents [4, 6, 6, 6, 7, 7]

[Agent 0] Has comm info from agents [1, 2, 3, 4, 5, 6, 7]. Using it for navigation.

[Agent 0] Large turn detected ($\Delta v=1.82$). (-0.20,0.96) -> (-0.45,-0.85)

Meaning:

- Agent 0 drastically changed direction, likely avoiding potential congestion or reacting to others

[Agent 1] Received messages from agents [2, 4, 6, 6, 7, 7]

[Agent 1] Has comm info from agents [0, 2, 3, 4, 5, 6, 7]. Using it for navigation.

Meaning:

- Agent 1 now knows where almost everyone is

Results

Network	Drop Rate	Delay (steps)	Communication Quality	Queue Size	Agent Behavior	Log Behavior	Result
Urban	5% (Low)	1–3 (Short)	High (fast + reliable)	Small	Smart, coordinated	Lots of messages, early navigation	Most organized
Suburban	10% (Medium)	2–6 (Medium)	Medium (OK but sometimes delayed)	Medium	Some confusion	Messages mid-way, moderate reasoning	Balanced
Rural	30% (High)	5–15 (Long)	Low (mostly delay or lost)	Huge	Blind, slow, isolated	Almost no comm → “Slowed down...”	Least organized