Simulation Aided Anticipatory Congestion Avoidance for Warehouses

Submitted a paper in IEEE CASE 2022 (Under Review)

Prepared by:

Hardik Bhati (IIT2019013)

Garvit Suri (IIT2019057)

Table of Contents

Commonly Used Words in the Report and their Meaning	**3**
Introduction	**4**
1.1 Warehouse Simulation	**4**
1.2 Importance/ Relevance of the Problem	**4**
1.4 Applications of Warehouse Simulation	**4**
1.5 Major Challenges in this Domain	**4**
Literature review	**5**
3. Warehouse Simulator	**6**
3.1.1 Product Perspective	**6**
3.1.2 Product Functions	**6**
3.1.3 Operating Environment	**7**
3.1.4 Principal Actors	**7**
3.1.5 User Documentation	**7**
4. Phases of Simulation	**8**
5. Software Outline	**10**
6. Modelling (Intelligent Driver Model)	**15**
6.1 Microscopic Models	**15**
6.2 Mathematical Model	**15**
6.3 Working of IDM	**16**
7. Methodology	**19**
7.1 Making Simulator continuous	**19**
7.2 Order Pickup Schemes	**19**
7.3 Collision Avoidance	**20**
7.4 Intersection Management	**20**
7.5 Path Planning	**24**
7.5.1 Abstracting the Graph	**24**
7.5.2 Calculating the Path	**25**
7.5 Congestion Avoidance	**26**
8. Architecture and Pseudo Codes	**31**
8.1 General Architecture	**31**
8.2 Behavioural Finite State Machine (BFSM)	**32**
8.3 Pseudo Code for GOTO GOAL Controller	**33**
8.4 Block Diagram for Agent Handler	**35**
8.5 Pseudo Code for Agent Handler	**36**
9. ER Diagram	**37**
10. Results	**38**
11. Conclusion	**41**

Commonly Used Words in the Report and their Meaning

Word	**Meaning**
Warehouse	Virtual Diagram of actual Warehouse
Rack	Container which can hold many items
Intersection	Meeting Point for two or more Roads
Task	Work Assigned to Bot
Goal	Many Goals make one Task
Ambient-Spaced Graph	Graph which contain every Point in Warehouse
Roadmap	Graph which contain every Intersection of Warehouse
Congestion	Too many bots nearby at a Point

1. Introduction

1.1 Warehouse Simulation

Our main aim is to simulate warehouse robots in a way such that they are able to successfully simulate a real working environment in a warehouse to significantly improve the delivery times for ferrying hundreds of millions of items ranging from guitar strings to car parts in real modern fulfilment centres like warehouses responsible for online marketing. This simulation involves robots managing tasks like stacking stocks, efficient order retrieval, compiling orders, sorting them and then passing them on to sorting bins all in symphony with small robots doing these tasks of transporting parcels within the centres with proper collision avoidance and congestion control mechanisms.

1.2 Importance/ Relevance of the Problem

Our problem has utmost importance and relevance in real life for improving overall efficiency and costs in online marketing. The substitution of automatons in place of dashing human workers can make a colossal difference to a warehouse company's bottom line. It's estimated that a fully automated warehouse can hold 50% more stock and retrieve that stock 3 times faster. This reduces the overall fulfilment cost by 40%. Cheaper, quicker warehouses mean products that are more affordable for the end-user and crucially products are much more likely to be on a van driving down the street the following day.

1.4 Applications of Warehouse Simulation

There are many applications for our warehouse simulation software, namely:

- Reducing overall fulfilment costs
- Much faster and quicker services to the end-user
- Less error-prone systems
- A model to test how to improve efficiency in real life in warehouses by modelling the same in our system like to store stock in randomised fashion or category wise.

1.5 Major Challenges in this Domain

Major challenges for us in this domain are:

- Optimal staffing or allocation of worker bots to task
- Figuring out the best algorithms for collision avoidance and intersection management avoiding deadlocks along the way.
- Modelling congestion management to get the best possible results out of our model
- Making the model robust to error-prone probable conditions so as to be able to scale the model out in the real world

2. Literature review

The problem of congestion has been addressed in the traffic systems since long. Our proposal is very similar to a recent work of Street et al. [12] who used probabilistic planning that synthesized robot behavior while considering congestion. With the help of PTD (phase-type distributions), they tried to construct a continuous-time Markov Chain (CTMC) that helps in computing the probabilities of bots at different times. They finally build a PRT (probabilistic reservation table) that is used as an input for the planning problem. We add the notion of online replanning to the general framework. Moreover, we strongly argue (and show by experiments) that for a fully autonomous system it is possible to accurately predict the congestions and speeds rather than assuming the same to be within assumed probabilistic bands.

Digani et al. [13] designed a Probabilistic Eulerian traffic model, where the authors proposed to partition agents into groups on the basis of the position and considered the whole groups for planning using centralized planning and after that treated the agents within a group using a decentralized system. The authors predicted the traffic flow which further helped in planning better paths. The model anticipates density and congestion, however, the anticipation is based on heavily parametrized and heuristic models. Our approach instead uses real data to convert density to speed to be used for planning, while the published plans are used in the same formulation to anticipate densities.

The ant-inspired trail-following algorithm [14] has been used to reduce congestion in multi-robot systems. The authors focused on building multiple-lane highways and then robots followed a lane-changing mechanism to get to the goal. Congestion occurs when many bots try to go to the same goal at the same time, causing congestion that affects the performance/throughput of the system. Marcolino and Chaimowicz [15] tackled this problem by using a distributed coordination algorithm. They used the concept of local sensing and communication and designed a probabilistic finite state machine through which robots are able to coordinate themselves and try to avoid the congestion situation. Srivilas and Cherntanomwong [16] designed a routing algorithm specific to warehouses with congestion considerations as per the current levels using the ACO algorithm. Such schemes can only avoid congestion after it has happened, while the proposed approach anticipates and avoids the congestion.

Lerman and Galstyan [17] studied a related system where robots collect pucks and deliver them home. The authors studied the effect of the number of robots on the system's overall performance and formalized the decrease in the performance of the individual robot as the number of robots increased. We instead study the same effect formally using the concept of fundamental diagrams. dos Passos et al. [18] solved the problem of congestion control for robotic swarms to maximize the throughput of the target area. The authors proposed a SQF (Single Queue Former) in which a queue is formed to the target area and a TRVF (Touch and Run Vector Fields) in which the robots are made to touch the boundary of the circular area by using vector fields. The paper suffers from the assumptions like constant linear velocity and fixed distance between the bots, while the proposed work generalizes the flow to every segment of the network rather than just the target area.

The problem of anticipatory congestion avoidance for a fully automated warehouse system has still not been actively researched, which is the aim of the paper. Stress is laid on the learning the speed to density conversion and using the same to iteratively plan the path using density and computing the density using the plan. To mitigate errors, frequent re-planning is introduced. These together make a novel congestion avoidance mechanism not prevalent in the literature.

3. Warehouse Simulator

The simulator for the Warehouse plays a very important role in this Project. We have implemented the simulator in a way that is very much scalable and can account for a large number of orders or bots also.

3.1 Overall Description

3.1.1 Product Perspective

This simulator will provide the users with a platform to mimic all the functionalities in a smart Warehouse. Users can generate random orders, and the truck loading zone is there for getting new items in the warehouse. A functionality for sorting bots is also there where orders are sorted according to the Pincode to which it's directed.

3.1.2 Product Functions

Our simulator supports the following features which are listed below :

1) To generate a random order according to the wish of the user. We can also decide after how much time or by what frequency the new orders will come.
2) Similar to order, we can also decide after how much time or how frequent the Truck will come with items that are to be filled in racks in the Warehouse.
3) Simulation of bots inside the warehouse which is currently implemented using Shortest Path Algorithms (BFS and A-Star Algo).
4) Simulation of Sorting bots which is done using a conveyor belt, originating from Human Counters leads to the Sorting Area.
5) Number of Racks, Number of Human Counters, Number of Bots (of any type) or the area of Sorting Area everything can be modified according to the user's preference. Also, the speed by which our bots move can also be changed as per convenience.
6) For effortless tracking of events, every minute-to-minute detail is filed in the log file which makes it very easy to track.
7) The database is also linked to our Simulator so that users can view orders history, which item is stored in which rack and which bot is carrying which items. Primarily, making our Simulator scalable.

8) We can pause the whole simulation as per convenience for better debugging for the same in future.

3.1.3 Operating Environment

The simulator will be compatible with any Operating System which is compatible to run Python3 and have MongoDB installed.

3.1.4 Principal Actors

The Principal Actor in our simulator is one user only who decides the size of the warehouse, orders frequency, etc.

3.1.5 User Documentation

The User will be provided with a User Manual on how to operate the software along with terms and conditions which must be agreed to in order to proceed using the software.

3.2 Objectives

Our main objectives are listed below :

1) To have a robust system for the warehouse simulation where whenever a new order comes there is a fair allocation of orders to robots which will be responsible for that order only.
2) There should be no collisions between two bots or there should not be any location in our grid in which there are 2 or more robots at the same time.
3) There should not be any deadlock situation where few robots are jammed together and there is no progress with time in that region.
4) If in a situation of deadlock then we need to have detection measures and some mechanism to get out of the deadlock state.
5) There should not be any livelock situation as well where though the robots may seem to move or state change may be happening but there is no progress with time.
6) From the human Counter to the conveyor in the Sorting Area, the sorting bots should place the order into the intended dumping pit only with keeping in account of collision prevention.
7) Proper congestion management using heat maps.

4. Phases of Simulation

4.1 Stacking Phase:

- **Working:** The working is quite simple with a number of delivery trucks coming at frequent intervals to provide stock for the workshop and in this phase, stacking robots will be in a queue waiting for different stock to be handed over to them so they put it in their place in the workshop which is decided by a centralised system according to different criteria as needed for eg, randomised fashion, most popular ahead of the rest, category wise, etc. As soon as the stock reaches the compartments or racks, the item type and quantity gets updated in the database to be considered for future orders. The Stacking Phase is in commotion now and will be ready soon.

- **Stacking Bots:** These bots have the job to take the stock coming from delivery trucks and stack it into the warehouse compartments racks according to a defined order. Once these bots reach the queue up to the delivery place, they will get different items with their quantities specified to the bot carrying them. As soon as these bots get the items they get a signal to place them on a particular rack in the warehouse. And as soon as these bots make their way to the particular rack, the database gets updated and the item is available to be placed on the order list.

4.2 Rack to the Human Counter:

- **Working:** The working goes on something like this, as soon as an item is ordered from an automated order generator, the order goes on to list what item types and their quantities have been ordered and the whole order gets assigned a particular human counter and after that our centralised system in place checks how many racks are needed to complete the order. After figuring out all the racks needed, the nearest rack bot to the rack gets assigned and it first goes to the rack to pick up the rack and then goes to the human counter. When all of the racks reach the human counter, an order is marked in the sorting phase and the whole thing gets stuffed in a single box and is sent over the conveyor belt to the sorting section of our warehouse.

- **Rack Bots:** These bots mostly sleep through much until they get a mission to carry the rack it got assigned to the marked human counter. They first go to the rack and pick it up and then carry it to the human counter assigned. Once the rack is assigned to the bot, the database gets updated as that rack is unavailable for the time being and is not free for new orders for some time. As soon as the bot reaches the human counter with the rack, the database gets updated as to what part of the order reached the counter and what part is left still. After its work is done at the human counter, it goes back to the rack to keep the rack where it came from. It gets assigned to a new rack if any part of some order is pending to be assigned and it goes on the tiring journey again. If no such part of the order is left to be assigned, it quietly goes to sleep beneath the racks until it's called again.

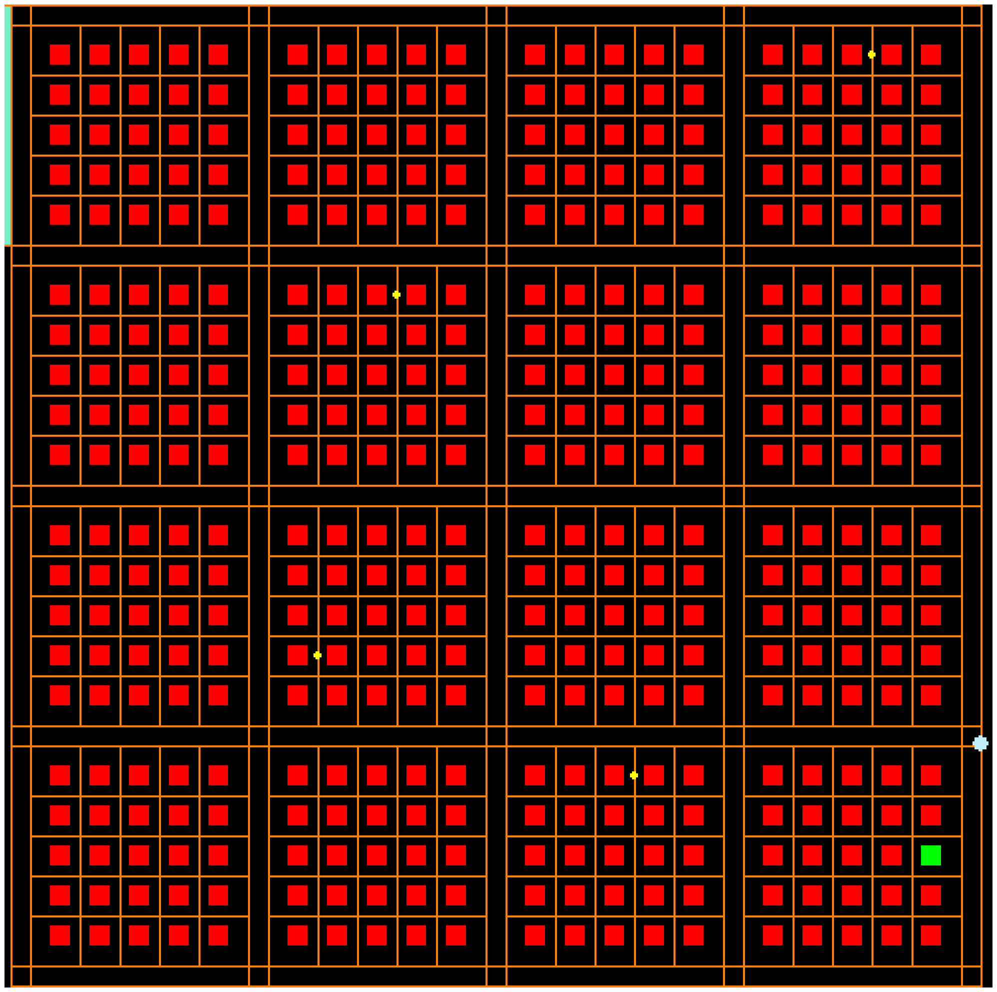
4.3 Sorting Phase:

- **Working:** The working is quite simple with every order which is marked in the sorting phase having to come to the sorting section after being put on the conveyor belt by their respective human counters. This phase acts out as its name states, it just sorts every parcel for an order it receives and checks its pin code and just assigns a sorting bot to dump the parcel into the suitable dumping bin for the respective Pincode. Once the order is dumped in the dumping bin, the database gets updated and the order is marked completed in the warehouse and the order is set to go in a particular delivery van which makes sure that the user who placed the respective order gets it.
- **Sorting Bots:** These bots have the job to take the parcel it gets from the sorting station and place and place the order in the dumping bin according to its pin code. Most of the sorting bots are lined up in a queue to get the parcel as soon as the sorting station receives it and some excess robots rest in a rest space, from where they get called up one after the other as soon as the sorting queue starts getting empty. Once they dump the parcel the database gets updated as mentioned above. After dumping the parcel, the sorting bots will go into the rest space where they will join the queue in hopes of getting more parcels in future to dump.

5. Software Outline

In this section, there will be a discussion about the layout of various portions of our software with the help of screenshots and brief explanations wherever required.

5.1 Item Storage Area in Warehouse

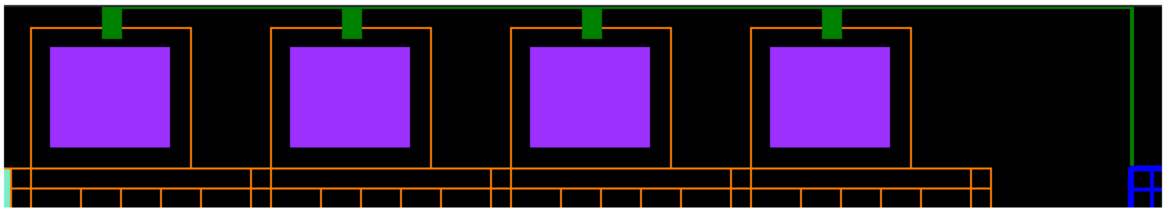


This section shows the various racks which are depicted in the picture using the following colour map :

Colour	Meaning
Orange Rack	Rack in Place
Green Rack	The rack is not at its place (A robot is carrying it)
White/Yellow/Red Circle	Moving Bots
Yellow Circle (of small size)	Bots that are currently idle

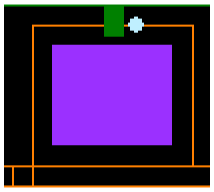
The yellow lines are our unidirectional roads which are in alternate fashion in such a way so that it's possible to reach from any rack to any other rack.

5.2 Human Counters



The purple boxes shown in the image above are the human counters which are responsible for packing once all the items of an order are received.

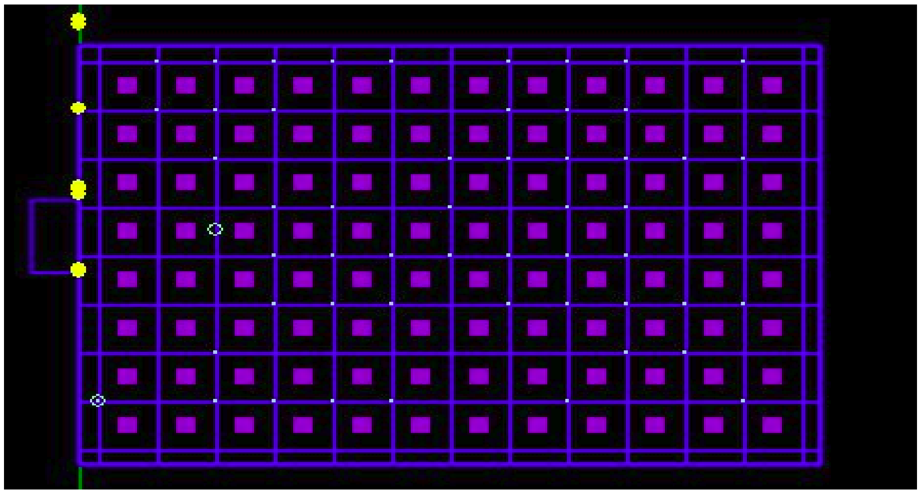
Whenever a robot wants to give the order to the human counter it enters through one endpoint and comes out through another. A robot delivering items to the human counter is shown below.



The green line here depicts that once for any arbitrary order all the items reach the human counter, there the human counter packs it and puts it over the conveyor belt which is the green lines emerging from each human counter which takes the packed order to the Sorting Zone.

5.3 Sorting Zone

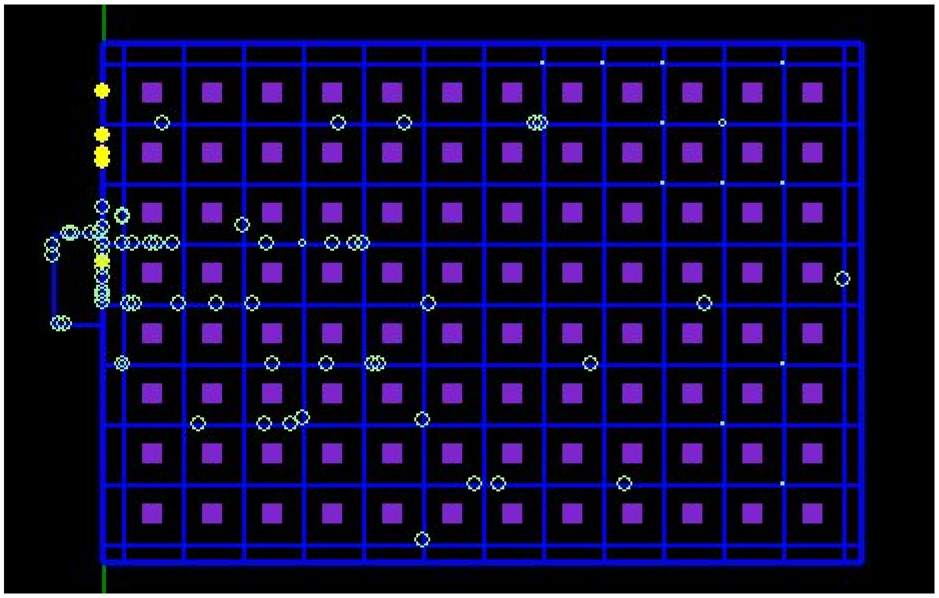
On the right-hand side of the screen lies the Sorting Zone of our Warehouse after which the orders are ready to ship to various places.



Colour Mapping for this section is shown below :

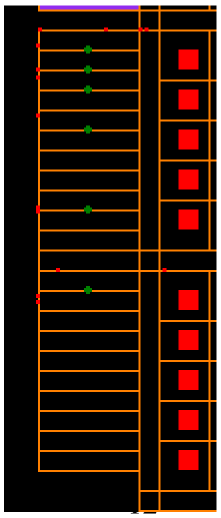
Colour	Meaning
Purple Square	Dumping Pits (acc to Pincode)
Yellow Circle	Packed Order is coming through Conveyor Belt
Light Blue Circle	Sorting Bot is moving an order
Light Blue Circle (of small size)	Sorting Bots which are currently idle

A sorting Bot in action is shown below:



The dark blue lines around the sorting zone are the one-way roads like it was in Item Storage.

5.4 Charging Zone:

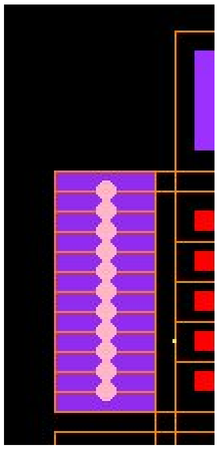


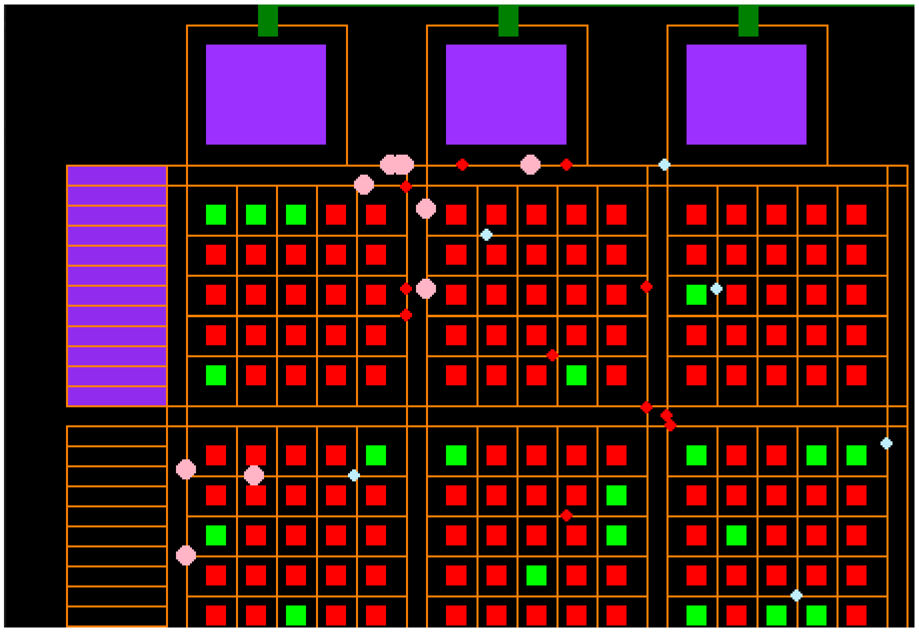
On the bottom left of our window, we have a charging station with multiple charging lanes to charge in. An agent with a charge less than the threshold goes over there by looping over the left of the charging station and going in the charging lane assigned to it.

Here the green agents are currently charging and the small red agents are the ones which are going to sit on their allotted lanes..

5.5 Truck Agents Zone:

On the top left of our window, we have our truck agents zone where truck agents spawn and load in items at spawn location and deposit them in racks told them to and again come back to spawn to load the workshop with items till the truck gets devoid of items.
Below is the screenshot for the spawn locations of truck agents which are pink and then delivering items to respective racks.



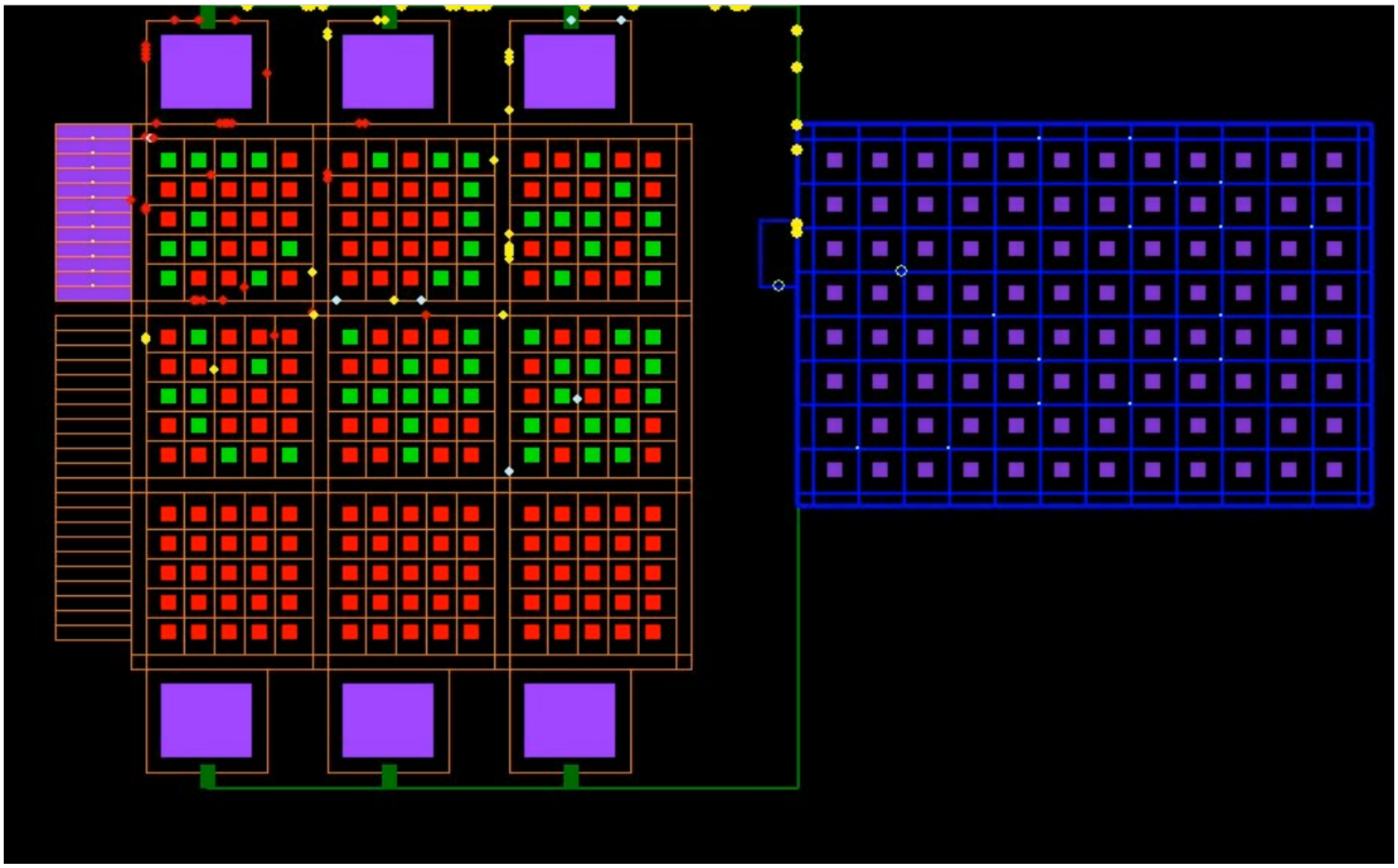


5.6 Warehouse Bots:

Now that we have applied Congestion Management, to get a better idea we have a dynamic color Scheme which is according to the Average Congestion around them. So, that color scheme is given below :

Colour	Meaning
White Circle	Low Congestion
Yellow Circle	Moderate Congestion
Red Circle	High Congestion

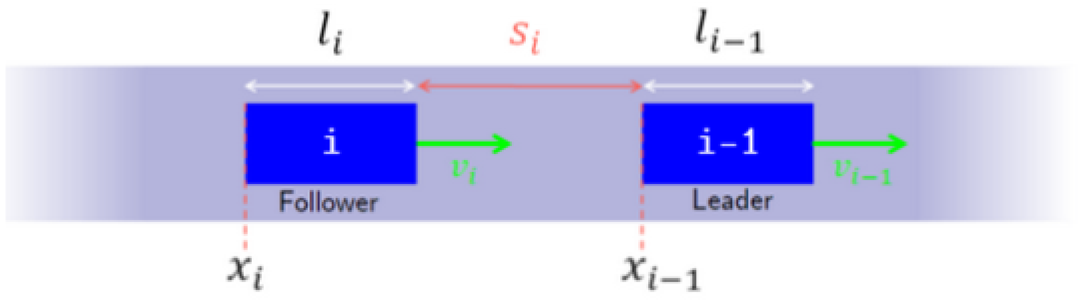
As shown below, we have the screenshot of our simulator running in action with Number of Bots=60 and Order are getting generated after every 10 seconds.



6. Modelling (Intelligent Driver Model)

6.1 Microscopic Models

We use this model to describe the behaviour of a single agent. As a result, every agent operates on its own using environmental inputs, to fit in a multi-agent system.



Every agent has its x_i(position), v_i(velocity), a_i(acceleration), l_i(length) and s_i(distance from agent ahead).

Furthermore, $s_i=x_i-x_{i-1}-l_i$ and $\Delta v_i=v_i-v_{i-1}$

6.2 Mathematical Model

Using the Intelligent Driver Model developed in 2009 by Hennecke et al[3], we use the acceleration of the *i*-th agent as a function of its parameters and those of the agent in front of it. The dynamics equation is defined as:

$$\frac{dv_i}{dt} = a_i\left(1-\left(\frac{v_i}{v_{0,i}}\right)^{\delta}-\left(\frac{s^*(v_i,\Delta v_i)}{s_i}\right)^2\right)$$

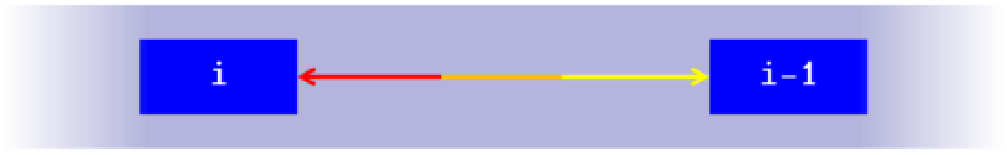
$$s^*(v_i,\Delta v_i) = s_{0,i}+v_iT_i+\frac{v_i\Delta v_i}{\sqrt{2a_ib_i}}$$

The other parameters are:

$dv_i/dt = a_i(1-(v_i/v_{o,i})^{\delta}-(k/s_i)^2)$

- **s_{0i}** : is the minimum desired distance between the vehicle *i* and *i-1.*
- **v_{0i} :** is the maximum desired speed of the vehicle *i.*
- **δ :** is the acceleration exponent and it controls the “smoothness” of the acceleration.
- **T_i :** is the reaction time of the *i*-th vehicle’s driver.
- **a_i :** is the maximum acceleration for the vehicle *i.*
- **b_i :** is the comfortable deceleration for the vehicle *i.*
- **s*** : is the actual desired distance between the vehicle *i* and *i-1.*

s* comprises three terms.



$$s^*(v_i, \Delta v_i) = s_{0,i} + v_i T_i + \frac{v_i \Delta v_i}{\sqrt{2a_i b_i}}$$

- **s_{0i} :** minimum desired distance.
- **v_iT_i :** is the reaction time, in our case its the time it takes our sensors to identify some change in configuration of environment and is used as a safety distance. It is the distance the agent travels before it reacts.
  Since speed is distance over time, distance is speed times time. Hence, v_i*T_i
- **$(v_i\ \Delta v_i)/\sqrt{(2a_i\ b_i)}$:** It's a speed-difference-based safety distance. It represents the distance it will take the vehicle to slow down (without hitting the vehicle in front), without braking too much (the deceleration should be less than **b_i**).

6.3 Working of IDM

Agents are assumed to be moving along a straight path and follow the equation:

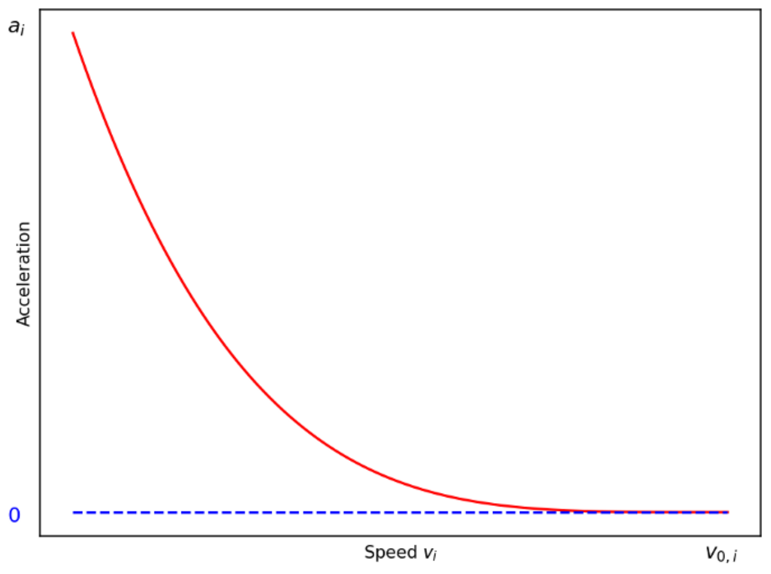
$$\frac{dv_i}{dt} = a_{free\ road} + a_{interaction}$$

$$\begin{cases} a_{free\,road} = a_i\left(1 - \left(\frac{v_i}{v_{0,i}}\right)^{\delta}\right) \\ a_{interaction} = -a_i\left(\frac{s^*(v_i, \Delta v_i)}{s_i}\right)^2 \end{cases}$$

We have a **free road acceleration** and an **interaction acceleration**.

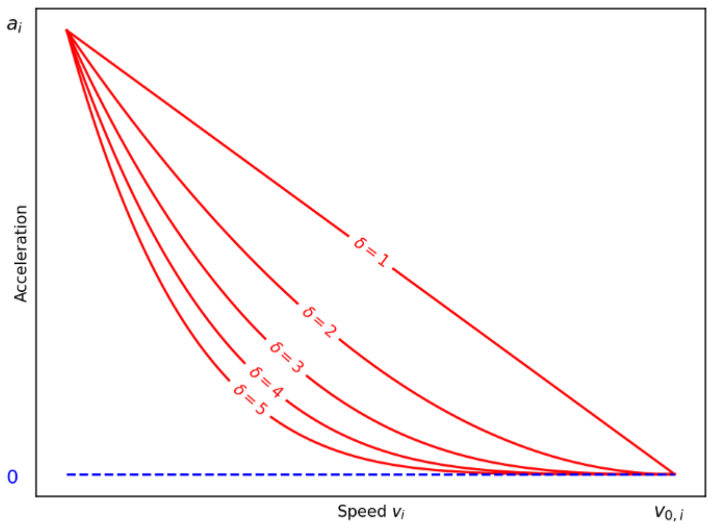
$$a_{free\,road} = a_i\left(1 - \left(\frac{v_i}{v_{0,i}}\right)^{\delta}\right)$$

The **free road acceleration** is the acceleration on a free road, that is, an empty road with no agents ahead. Plotting the acceleration as a function of speed **v_i** we get**:**



When the agent is stationary (**v$_i$=0**) the acceleration is maximal. When the agent speed approaches the maximum speed **v$_{0i}$** the acceleration becomes 0. This indicates that the **free road acceleration** will accelerate the agent to the maximum speed.

If we plot the v-a diagram for different values of **δ,** we notice that it controls how quickly the driver decelerates when approaching the maximum speed. Which in turn controls the smoothness of the acceleration/deceleration/



$$a_{interaction} = -a_i \left(\frac{s^*(v_i, \Delta v_i)}{s_i} \right)^2 = -a_i \left(\frac{s_{0,i} + v_i T_i}{s_i} + \frac{v_i \Delta v_i}{2 s_i \sqrt{a_i b_i}} \right)^2$$

The **interaction acceleration** is linked to the interaction with the agent in front. Following situations arise:

- **On a free road (s_i >> s*):**
  When the agent in front is far away, that is the distance $\mathbf{s_i}$ is dominates the desired distance **s***, the interaction acceleration is almost 0.
  This means that the agent will be governed by the free road acceleration.

$$\frac{dv_i}{dt} \approx a_{free\,road} = a_i\left(1-\left(\frac{v_i}{v_{0,i}}\right)^{\delta}\right) \quad ; \quad \left(\frac{s^*(v_i,\Delta v_i)}{s_i}\right)^2 \approx 0$$

- **At high approach rates (Δv_i):**
  When the speed difference is high, the interaction acceleration tries to compensate for that by braking or slowing down using the $\mathbf{(v_i \Delta v_i)^2}$ term in the numerator but too hard. This is achieved through the denominator $\mathbf{4b_is_i^2}$.

$$a_{interaction} = -a_i\left(\frac{s_{0,i}+v_iT_i}{s_i}+\frac{v_i\Delta v_i}{2s_i\sqrt{a_ib_i}}\right)^2 \approx -\frac{(v_i\Delta v_i)^2}{4b_i{s_i}^2}$$

- **At small distance difference (s_i << 1 and $\Delta v_i \approx 0$):**
  The acceleration becomes a simple repulsive force.

$$a_{interaction} = -a_i\left(\frac{s_{0,i}+v_iT_i}{s_i}+\frac{v_i\Delta v_i}{2s_i\sqrt{a_ib_i}}\right)^2 \approx -a_i\frac{\left(s_{0,i}+v_iT_i\right)^2}{{s_i}^2}$$

7. Methodology

7.1 Making Simulator continuous

We changed our simulator from a discrete system to a continuous one by using IDM and introducing real life parameters including velocity and acceleration.
We made roads in our simulator, each having information about the 2 intersections it connects and all the agents currently on that road.
Next, looping on each road, and further looping on every agent it currently has, each of whom updates their position seeing the agent ahead of it. The agent ahead of the pack has no bounds until the intersection is at some particular fixed distance after which it has to adhere to the Intersection Management, on how to update its parameters further.

7.2 Order Pickup Schemes

Scheme 1: This is the classical approach which Amazon does. In this, a mobile bot carries whole rack on it's back to the human counter and then human counter picks up the items from the shelf, then the bot places the rack back at it's place.

Scheme 2: We propose a new scheme in which the bot (re-designed) can now pickup the item from shelf of the rack directly. This approach converts the automation problem to classic Travelling Salesman Problem. This approach is expected to perform better than the former scheme because one potential method of improving performance in this is due to the fact that now we have boiled down the problem to TSP problem. Ordering of items to pick-up can affect optimality here.

Comparison with Example:

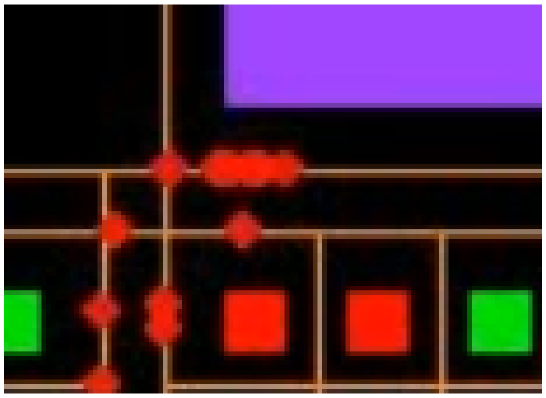
If there were 5 racks to which a bot needed to go to pick up items for a single order.

In Scheme -1, it will have to do 5 rounds to the human-counter back and forth. Travelling Distance will be more. On the brighter side, the load is shared between the 5 agents in this scheme.

In Scheme -2 , one single bot will go and pickup items from all 5 of the racks. After pickiing up all the items , it gets to the human counter only once.

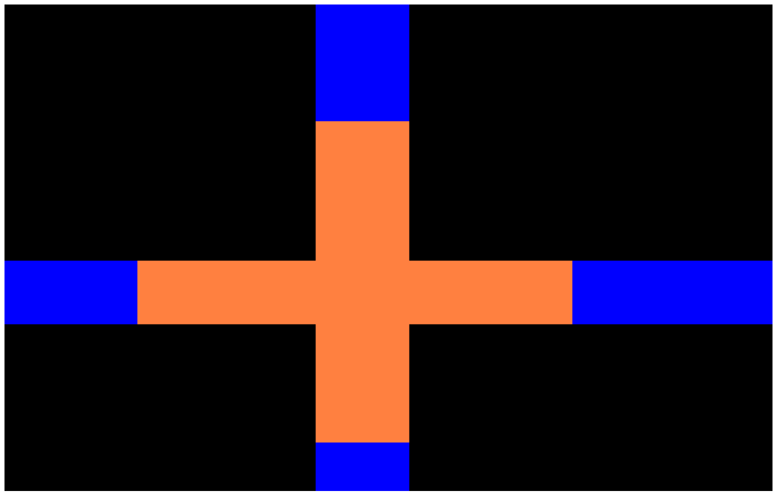
7.3 Collision Avoidance

Intelligent Driver Model takes care of Collision Handling leaving the intersections part. We use intersection management to modify IDM according to our simulator.
Every agent knows the parameters of the agent in front of it, and it successfully and robustly adjusts its parameters to maintain a safe distance with a good tradeoff between speed and safety.



7.4 Intersection Management

We had to incorporate Intersection Management in our model using IDM which assumes only straight roads not accounting for intersection.
Firstly, for incorporating Intersections in our model we moved from our previous approach where we just had a single point for intersection and changed it to a range of values to encompass the agent which now has length itself.



Now, whenever our agent reaches this intersection, we lock the intersection up for further use, and any other agent can't enter further in this intersection and treats this intersection as a big agent itself and IDM helps to control parameters.
Since agents do not stop suddenly, we had to check for every last agent in roads to check for distance from the intersection and once it reaches a threshold it books the Intersection for itself if it isn't booked and if it is booked the agent treats the intersection as an obstacle.

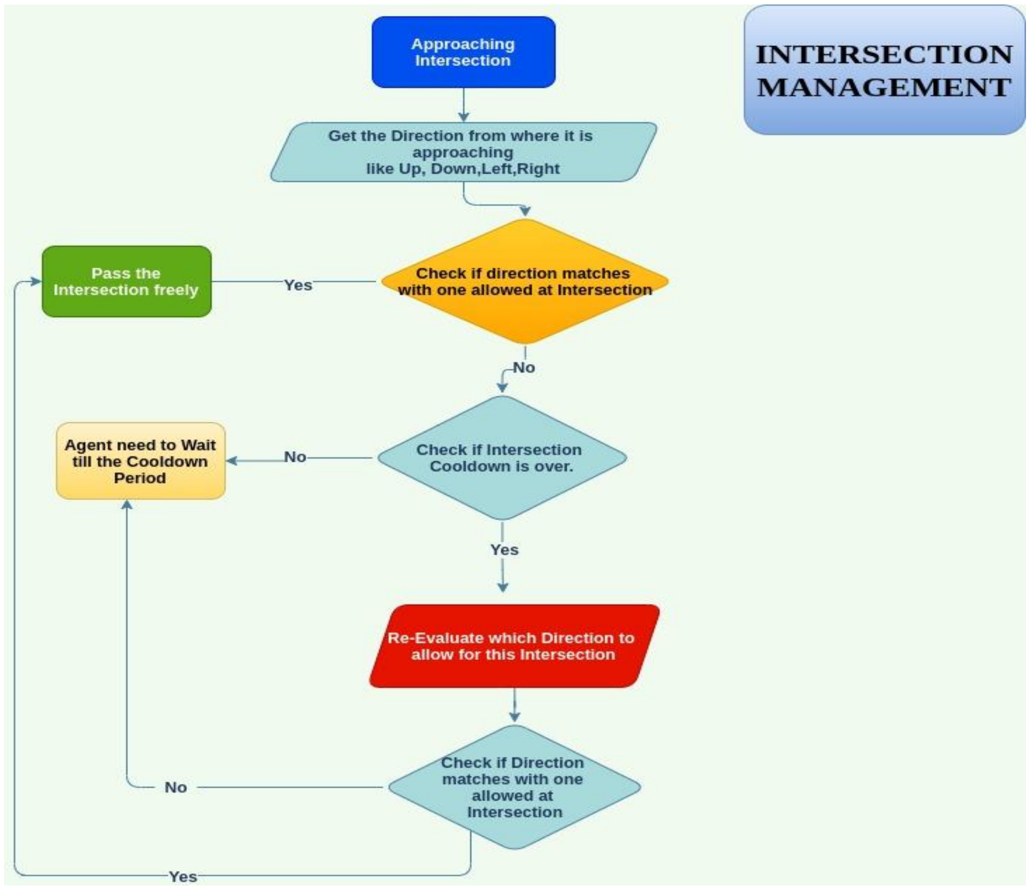
Now, for managing intersections for booking conditions,, we tried different approaches and the one that gives us the best results in different scenarios, that scheme is finalised.
To achieve Intersection Management, we visualised this problem as a Resource Allocation Problem in OS. We will try to do something similar to achieving Process Synchronisation using semaphore.

The only thing to worry about in any scheme we apply is the Deadlock situation, so we need to account for that also.

Entities playing important role in Intersection Management:

1) **Intersection Gateway** - It's a dictionary for all Intersections which is maintained by the Intersection Manager. This is like a lookup table for our Manager whenever he wants to respond to any query or update the table.
2) **Intersection Manager** - The intersection Manager is the main actor in our Intersection Management System. Any agent, who wants to pass an Intersection needs to ask the Intersection Manager first, and it replies with a response depending on which lane is allowed to go. This is very similar to **Query-Response Mechanism.**
   Intersection Manager is also the one using Intersection Management Policy and if an Agent demands it to update the lane at this Intersection then the Manager will decide the next lane who will be given turn by using the Policy and updates them in the Intersection Gateway Dictionary.
   Any Agent cannot call the Intersection Manager whenever it wants a lane to be open, if the Manager changed the lane just now then the Agent will have to wait for some time to again ask the Manager to update the lane. Then once the required direction for the Agent is allowed then it passes freely.

BLOCK DIAGRAM FOR INTERSECTION MANAGEMENT


INTERSECTION MANAGEMENT
Approaching Intersection
Get the Direction from where it is approaching like Up, Down,Left,Right
Check if direction matches with one allowed at Intersection
Yes
Pass the Intersection freely
No
Check if Intersection Cooldown is over.
No
Agent need to Wait till the Cooldown Period
Yes
Re-Evaluate which Direction to allow for this Intersection
Check if Direction matches with one allowed at Intersection
No
Yes

We used 2 different Schemes for Intersection Management which are as follows:

- **Scheme 1 (Normal Booking Mechanism)** - In this scheme, to decide on the priority for the lane to open, we have a booking mechanism. The agent which first books the intersection(there is a threshold on when can an agent book and it has to be the last agent on the road only) is only allowed to pass, and upon reaching the intersection range, we force the location of agent to be of centre of intersection and then the agent checks if the road the agent wants to go next to has any agent on it which is less than a particular threshold, if all conditions are met, then the agent is passed on to the road and booking is removed and we force the velocity of the agent to become zero if the agent has to make a turn at intersection otherwise it stays the same as when it entered the intersection.
  There is a chance of starvation in it, like in a case where one lane agents are waiting for a long period of time if for every time we are checking count the other lane is getting a chance.

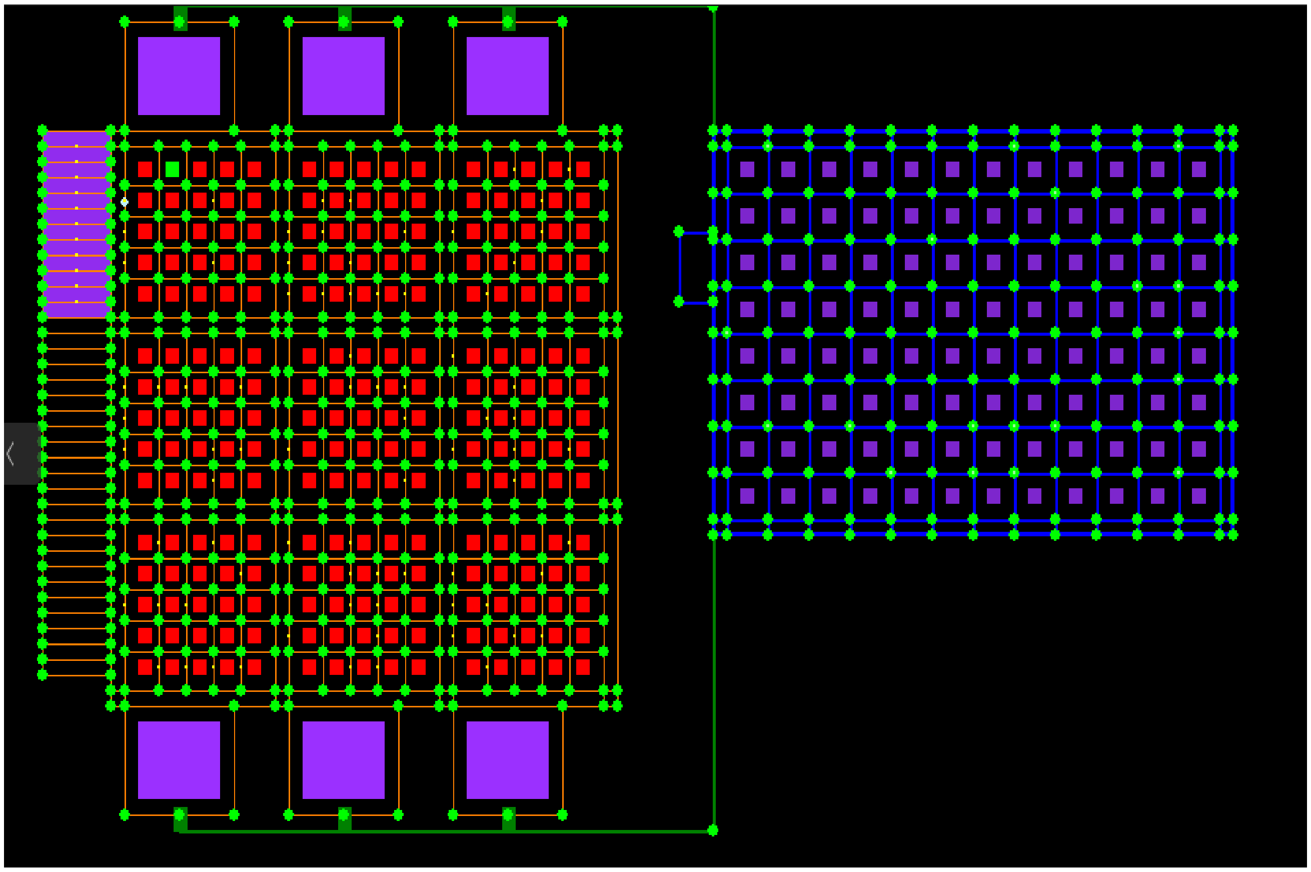
- **Scheme 2(Traffic Light)-** In this scheme, there are traffic lights which change according to some particular scheme which we discuss later, but for the time being, we are just changing it to a fixed rhythm. These lights allow only motion from one direction to the intersection but the agent once in the intersection is free to go anywhere it wants to go (ony where the road is directed to). In here also, since any agent can't just stop all of a sudden if the traffic light changes, here also there is a booking mechanism but it is restricted to only allow booking for a particular direction according to the traffic light. If the light changes and the booking is in some other direction, we allow that particular agent to pass even though that direction is not allowed according to the traffic light. But just as it passes, only the traffic light allowed direction is allowed for booking.

7.5 Path Planning

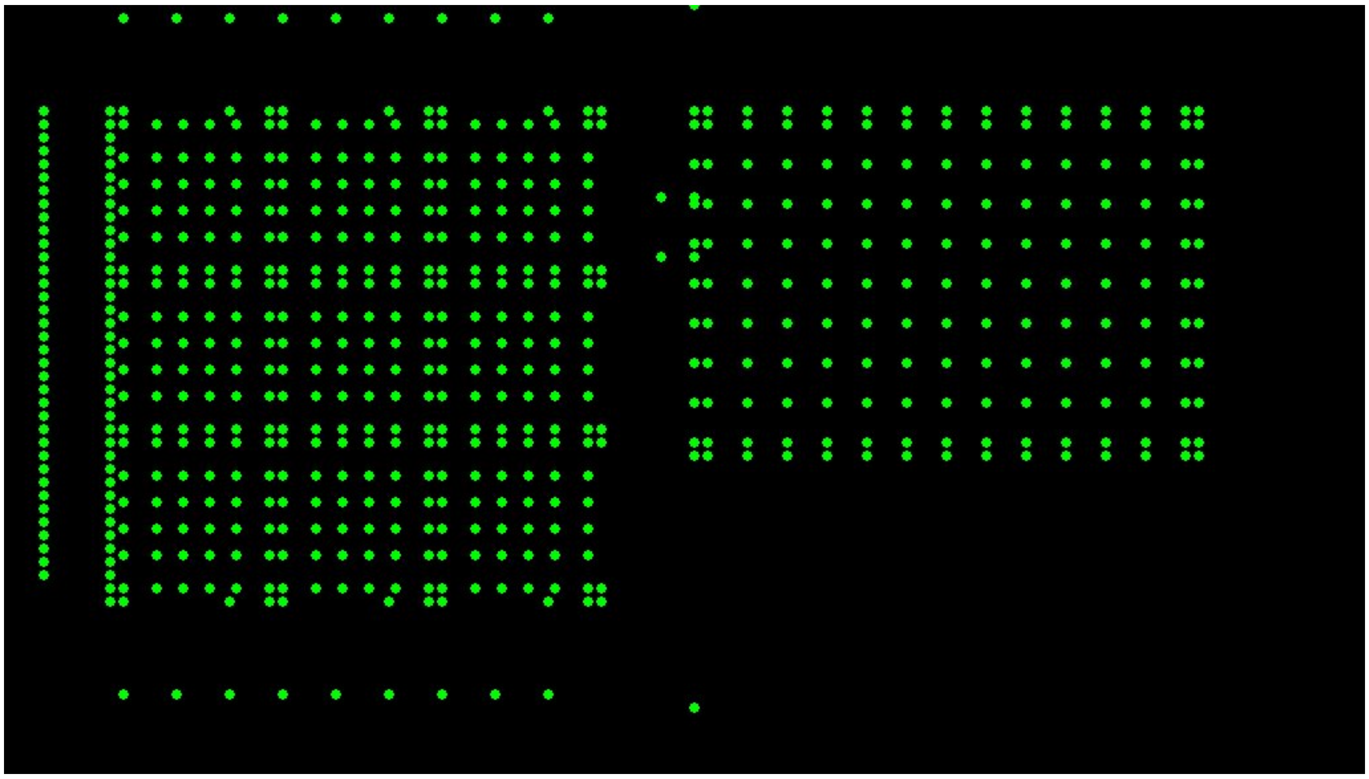
7.5.1 Abstracting the Graph

It was not possible to make the model in real-time by using the same graph for calculating the path and showing the agents on.
So we abstracted out the details and we made a **roadmap** considering directional constraints using DFS on a **workspace connectivity graph**.

Workspace connectivity Graph:



Roadmap:



7.5.2 Calculating the Path

To calculate the path between any point to a goal, we follow three main steps which are as follows:

1) **Access Point -** From the point where we are starting the journey, it is not necessary that the source is in our Roadmap. So in our Grid, we need to find the nearest Point to the Source which is in the Roadmap and that point would be the nearest intersection to that Point. So, using BFS we do the task of finding the access point for our journey.

2) **Connectivity -** On reaching the Access Point, we apply A* Algorithm using heuristic as Manhattan Distance from source to Goal when not using congestion. Now we will get a Path but in the Roadmap domain so we need to convert the Path to Workspace Connectivity Graph to move the agent. After this Phase, we will either reach the Goal or a point in the Roadmap Graph that is nearest to Goal.

3) **Depart Point** - If the Goal we are planning for is not in the Roadmap Graph, then we need to again find the simple straight path to the Goal.

7.5 Congestion Avoidance

If all the robots use the shortest path algorithm to go from the source to goal, it is evident that certain areas would be prone to congestion leading to extremely large travel times. It is therefore necessary that the routing algorithms avoid the congregation of a large number of robots at any specific place. This is done by routing the robot to take roads that minimize travel time and also minimize anticipated congestion. Our preference was to make a congestion management mechanism that does not depend upon the parameters. Furthermore, the application is fully controllable as the only entities in the system are the robots, which could be used for an accurate anticipation that is rarely possible otherwise for applications widely studied in research.

The path planning algorithm uses the roadmap to perform a graph search and give a route from the current position to the goal.

Let <u,v> be a directed road from intersection u to intersection v. Let T(u) be the time when the vehicle is anticipated to reach the intersection u as per the current search tree. The search aims to minimize the total travel time from source to goal. The cost of an edge <u,v> is thus given by (3).

$$w(u,v) = \frac{d(u,v)}{speed\left(\kappa\left(u,v,T(u)\right)\right)} + turn(u,v). \qquad (3)$$

Here d(u,v) is the distance between the intersections u and v. κ(u,v,t) is the anticipated density of the edge <u,v> at time t, speed(e) is the expected speed at density e and turn(u,v) is the time required to make a turn at intersection u, if needed, to orient towards intersection v computed by a knowledge of the maximum angular speed. The prospective time to reach vertex v from vertex u can be computed by (4).

$$T(v) = \begin{cases} \min\limits_{u \in \eta(v)} T(u) + w(u,v) & v \neq current\ position \\ 0 & v = current\ position \end{cases}. \qquad (4)$$

Here η is the neighbourhood function. The aim is to find the minimum time to reach the goal T(Goal) which is a graph search problem given the costs.

The first challenge in the search is the density to average velocity function speed() that can be obtained from the fundamental diagram. We ran millions of timesteps of our simulator to get the desired mapping from density to velocity for any given road. Once we had the data after simulation, we applied curve-smoothing to it to get a smooth curve for density v/s speed. Using the curve, for any density, we could have the value of the average velocity of bots for that road. The second challenge is the density estimation function κ(u,v,T(u)). This is done by two strategies, assuming stationary densities (strategy 1) and anticipating densities (strategy 2).

- **Strategy 1 (Stationary Densities)**

This strategy plans on a map assuming that the road densities shall remain stationary. While planning, we consider the densities of roads for future timesteps as the current density of that particular road, giving the cost function (5), where κ(u,v,t) is the density of edge <u,v> at current time tcur that can be observed from the occupancy map.

$$w_{strategy1}(u,v) = \frac{d(u,v)}{speed(\kappa(u,v,t_{cur}))} + turn(u,v). \quad (5)$$

The densities will change when the robot follows the path, and it is possible that several robots see a sparsely occupied road and travel to it at the same time causing congestion. To cater to that we introduce re-planning. Whenever any bot reaches an intersection, it does re-planning in hope of getting a better path than the already calculated one. We have a dense network of roads with many intersections causing frequent re-plannings.

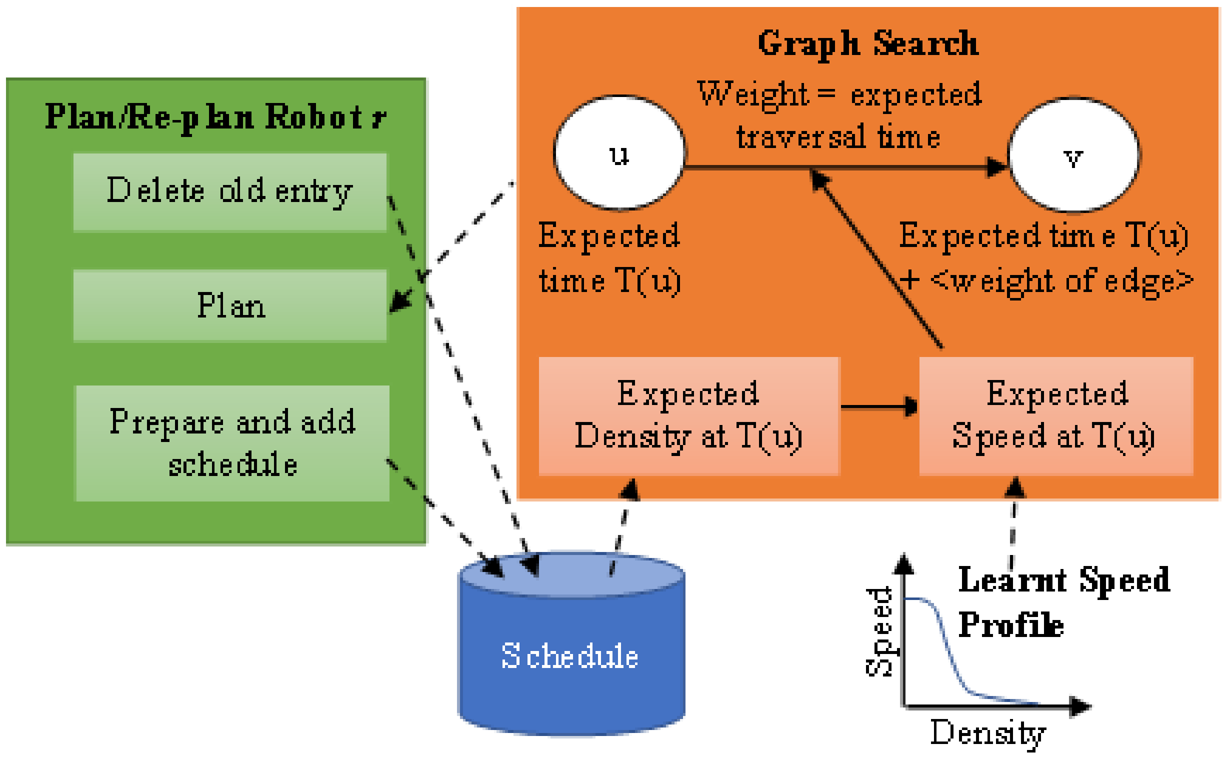


Figure 1. Congestion Avoidance. Schedule manages time-stamped trajectories for robots, maintained as the robots (re-)plan. It is used to anticipate densities/travel times from a learnt function for planning.

- **Strategy 2 (Predicting Densities)**

In this strategy, we predict the traffic at a future time and use the predicted density for planning with the cost function given by (3). Typically, it is not possible to accurately predict the density for a non-recurrent traffic scenario. However, we benefit from the fact that all the traffic entities are automated and controllable, or every robot publishes its route actively. We also fairly accurately know the average time it takes for any robot to pass any road in the route. The predicted density is calculated by timestamping each road with multiple bots and the time periods in which they pass a road, which is used for figuring out the number of agents at a road at a given time. The path is replanned at regular every intersection to give a better shot at figuring out correct trajectories of the agents and thereby getting a better time for the system. The system is shown in Fig. 5.
Let ς(u,v)={<t1,t2,r>} be the schedule of robots for road <u,v> as a set of entries, each entry denoting that the robot r shall be in the road from time t1 to t2. The schedule ς will change frequently and should ideally be a function of time; however, to keep the notations clean the time attribute is omitted. The expected density is given by (6).

$$K(u,v,t) = \sum_{<t1,t2,r>\ \varepsilon\ \varsigma(u,v)} F(t,t1,t2)$$

The F-Function takes in three parameters t (The time at which we want to find the expected density) , t1 (the time at which bot r will enter the Road <u,v>) , t2 (the time at which robot r will leave the Road <u,v>). In the original Scheme -2 , F function is just the indicator function which can be given as:

$$F(t,t1,t2) = \begin{cases} 1 & t1 - buffer \leq t \leq t2 + buffer, \\ 0 & otherwise \end{cases}$$

Assume that the robot r needs to plan (or re-plan). The planning (or re-planning) is done in three steps.
Step 1: Delete the robot's entry in the schedule. Let the old plan of the robot be given by τ'r=τ'r0,τ'r1,τ'r2, … (if any). The updated schedule ς after deletion is given by (7), where ← is the assignment operator that overrides the old value with the new assigned value.

$$\forall_i(\varsigma(\tau'^{i-1}_r, \tau'^{i}_r) \leftarrow \varsigma(\tau'^{i-1}_r, \tau'^{i}_r) - \langle t_1, t_2, r\rangle) \,. \qquad (7)$$

Step 2: Plan a new path using the expected density function (6) and cost function (3). Let the new plan be given by $\tau_r = \left[\tau^0_r, \tau^1_r, \tau^2_r, \ldots\right]$ starting from the last intersection τ^0_r encountered by the robot.
Step 3: Add the robot's schedule to all the roads in the plan. The anticipated time to arrive at intersection τ^i_r is given by (8). The updated schedule is given by (9).

$$T_r(\tau_r^i) = \begin{cases} T_r(\tau_r^{i-1}) + \frac{d(\tau_r^{i-1},\tau_r^i)}{speed\left(\kappa\left(u,v,T_r(\tau_r^{i-1})\right)\right)} + turn(\tau_r^{i-1},\tau_r^i) & if\ i \neq 0 \\ t_{cur} & if\ i = 0 \end{cases} \quad (8)$$

$$\forall_i(\varsigma(\tau_r^{i-1},\tau_r^i) \leftarrow \varsigma(\tau_r^{i-1},\tau_r^i) \cup \langle T(\tau_r^{i-1}), T(\tau_r^i), r\rangle) \text{ . } (9)$$

- **Extended Strategy 2 (Predicting Densities)**

In Scheme -2, while doing Density Estimation we were trying to predict the density using the knowledge of the schedule of the bots that have already planned their path.

The problem in the indicator-function like prediction is the abrupt drop/rise in the density. We assume our problem to be binary that the bot is either there or it is not there. Instead, this can be improvised if we introduce the notion of probabilities into planning.

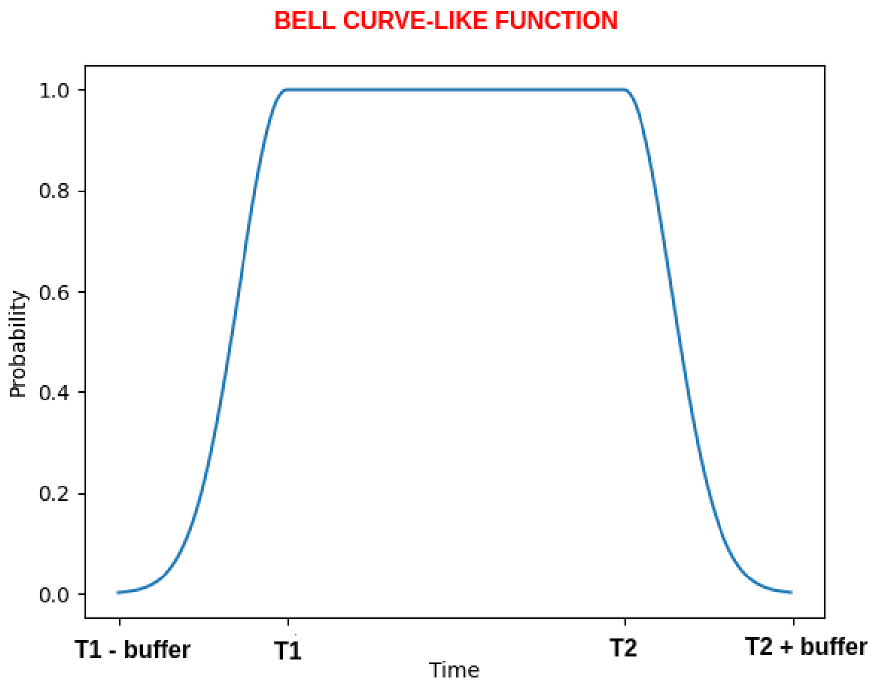
We achieve this by building over the original approach as:

- We know that if everything was ideal and deterministic, our bot will be on the particular road from T1 to T2. Therefore, we consider the probability of finding the bot on that road for the time interval T1 to T2 to be 1.
- For the time interval between [T1-buffer, T1) and (T2, T2+buffer], unlike the former approach, we consider it to be a decreasing exponential curve that approaches 0 at the extreme ends. We do this because as we move far from the intended time interval [T1,T2] , the probability must decrease intuitively.

The equation for the probability curve is given by:

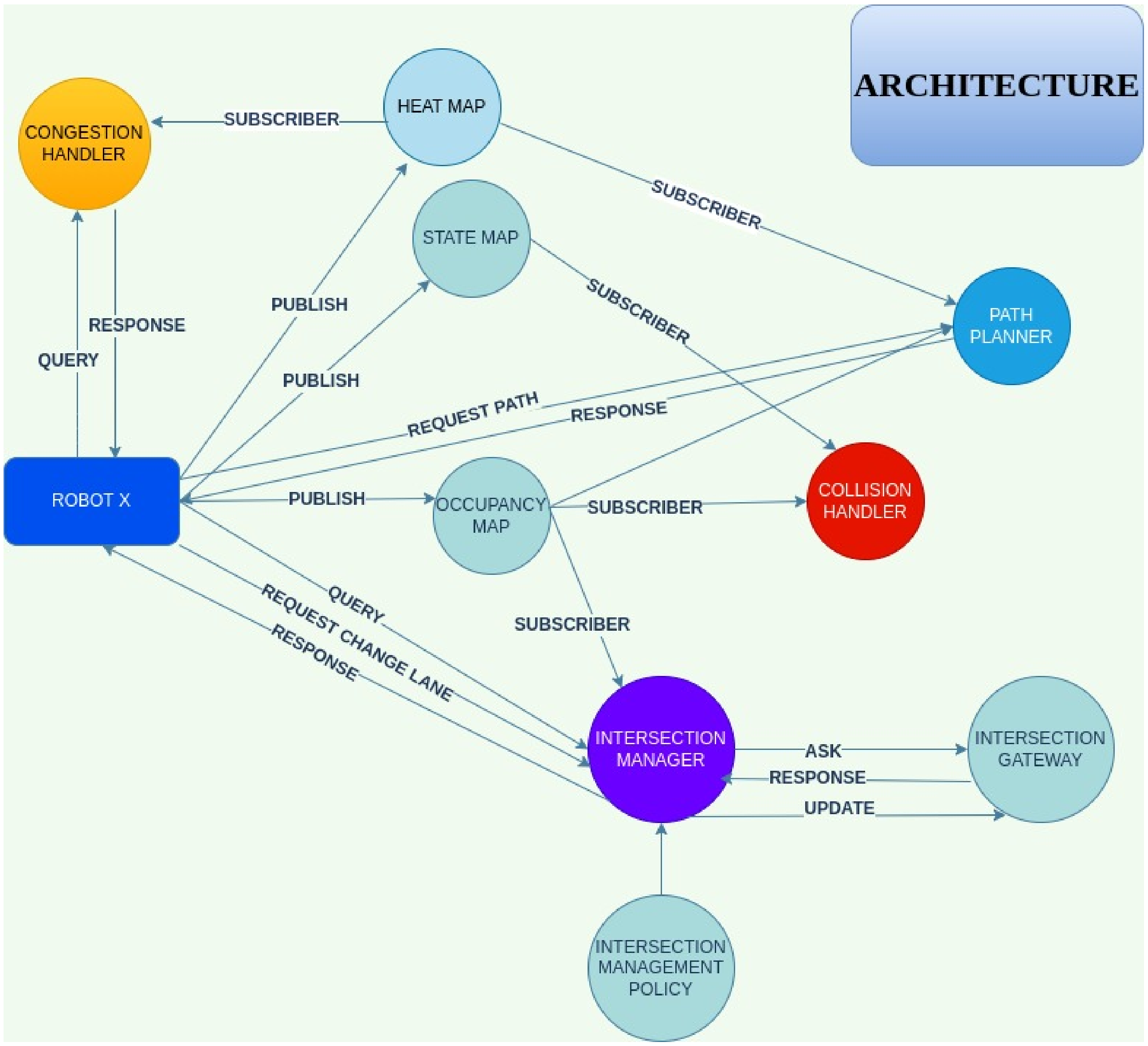
$$F_{NEW}(t,t1,t2) = \begin{cases} \exp\left(\frac{-(t-t1)^2}{buffer^2}\right) & t1 - buffer \leq t < t1, \\ 1 & t1 \leq t \leq t2, \\ \exp\left(\frac{-(t-t2)^2}{buffer^2}\right) & t2 < t \leq t2 + buffer, \\ 0 & otherwise \end{cases}$$

The probability plot is given by:



8. Architecture and Pseudo Codes

8.1 General Architecture

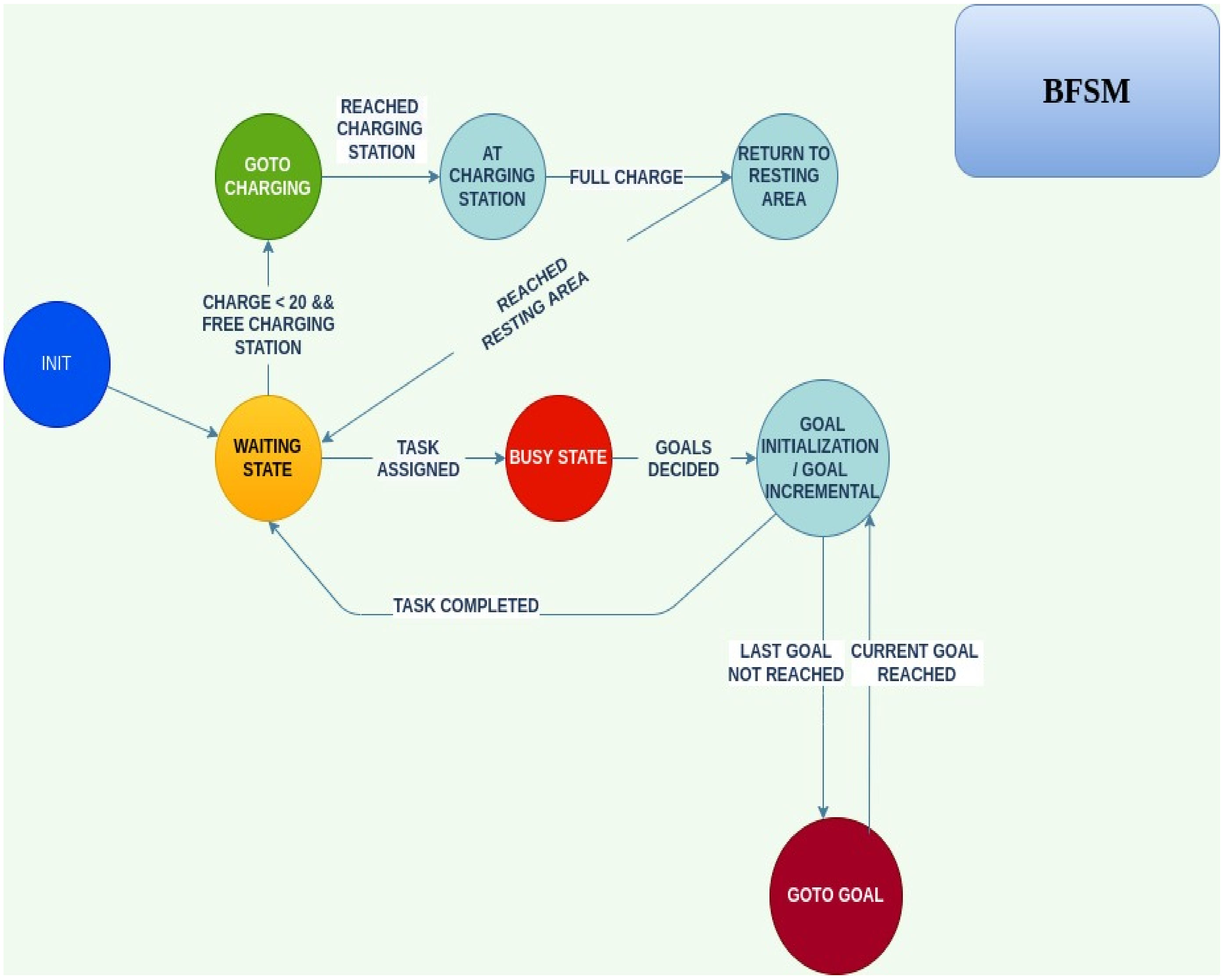


This diagram shows how various entities which were explained above interact with each other for any arbitrary Robot named X. Here we are mainly trying to show the dependencies between various entities also and their interaction is on which terms or conditions.
Broadly, there are two types of protocols that are happening in parallel that are: 1) Query and Response Protocol and 2) Publisher Subscriber Protocol.
Three new Terms in the diagram which are not explained above are as follows:

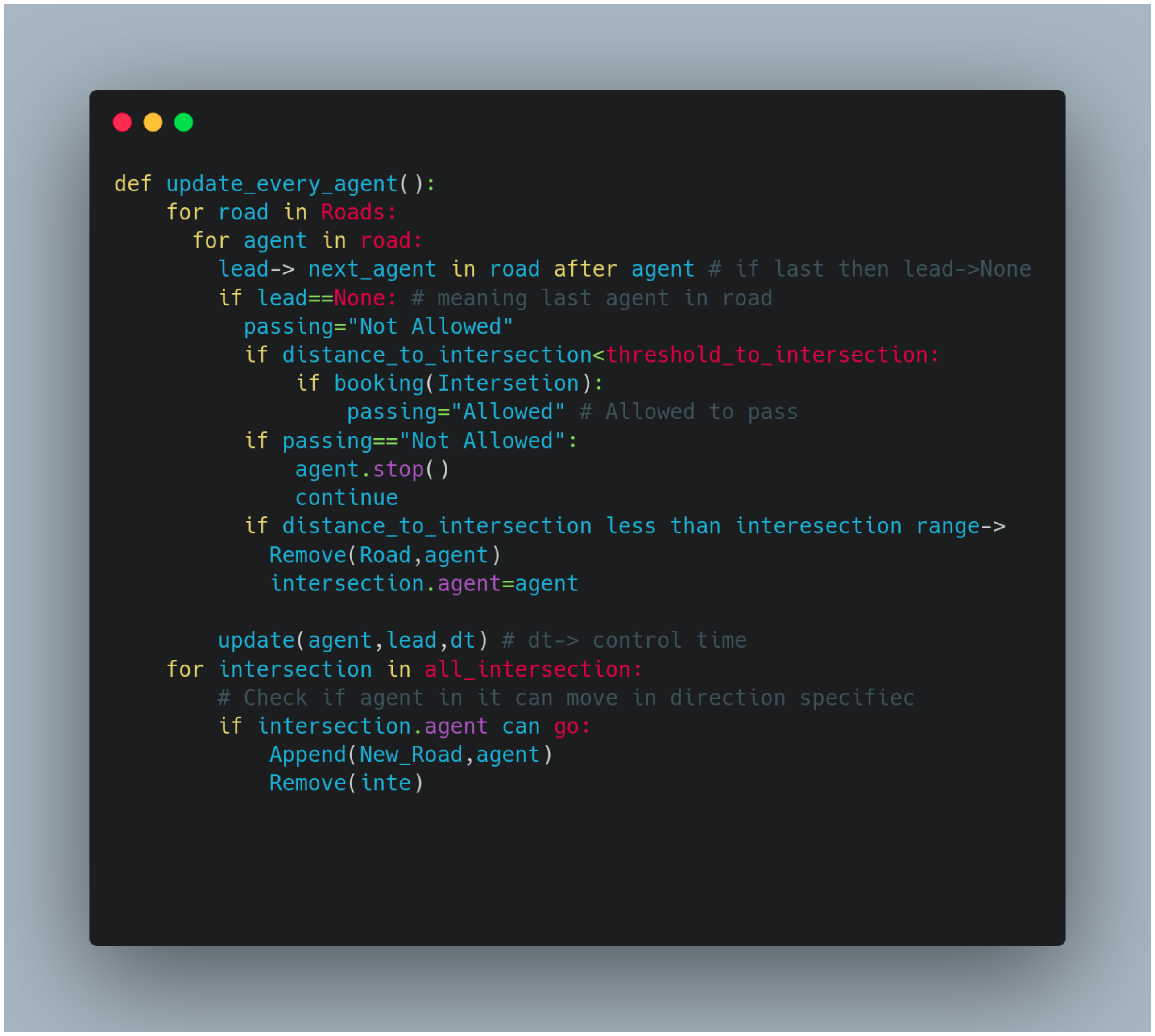
1) Occupancy Map - This dictionary tells us about which all Points are occupied by Bots.
2) State Map - This dictionary tells us about the states of various agents.
3) Heat Map - This dictionary tells us about the Heat Values at various positions of our Grid.

8.2 Behavioural Finite State Machine (BFSM)

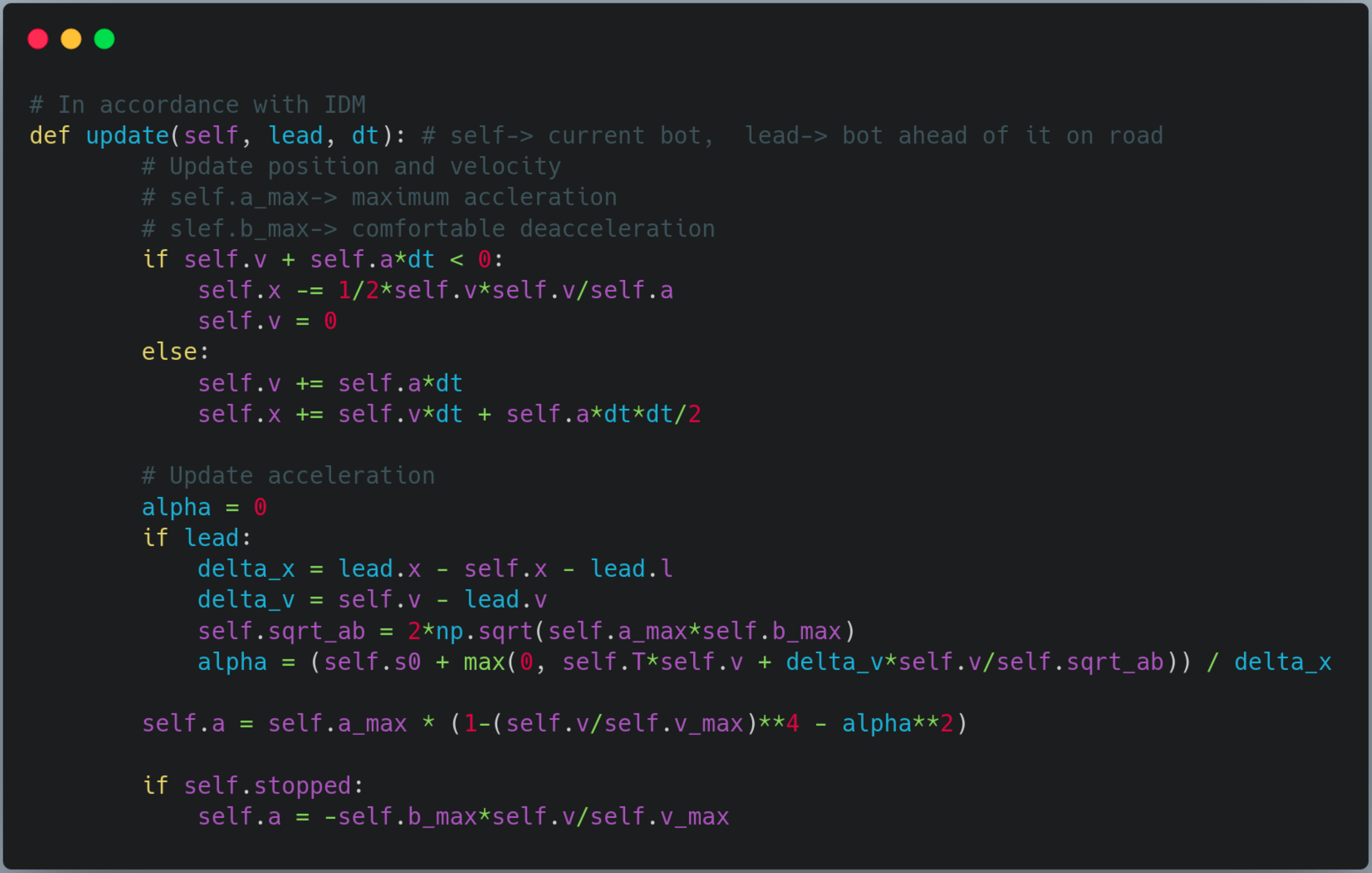
For modelling of our solution , we have made a simplistic BFSM for various states and to show how when a Task is given to the Agent, it is divided into various Goals and then completing all of them will result in completing a Task which would be done working in parallel with our Intersection Management and Congestion Management which is in our GOTO Goal controlled and would be explained in detail in the next part.



8.3 Pseudo Code for GOTO GOAL Controller

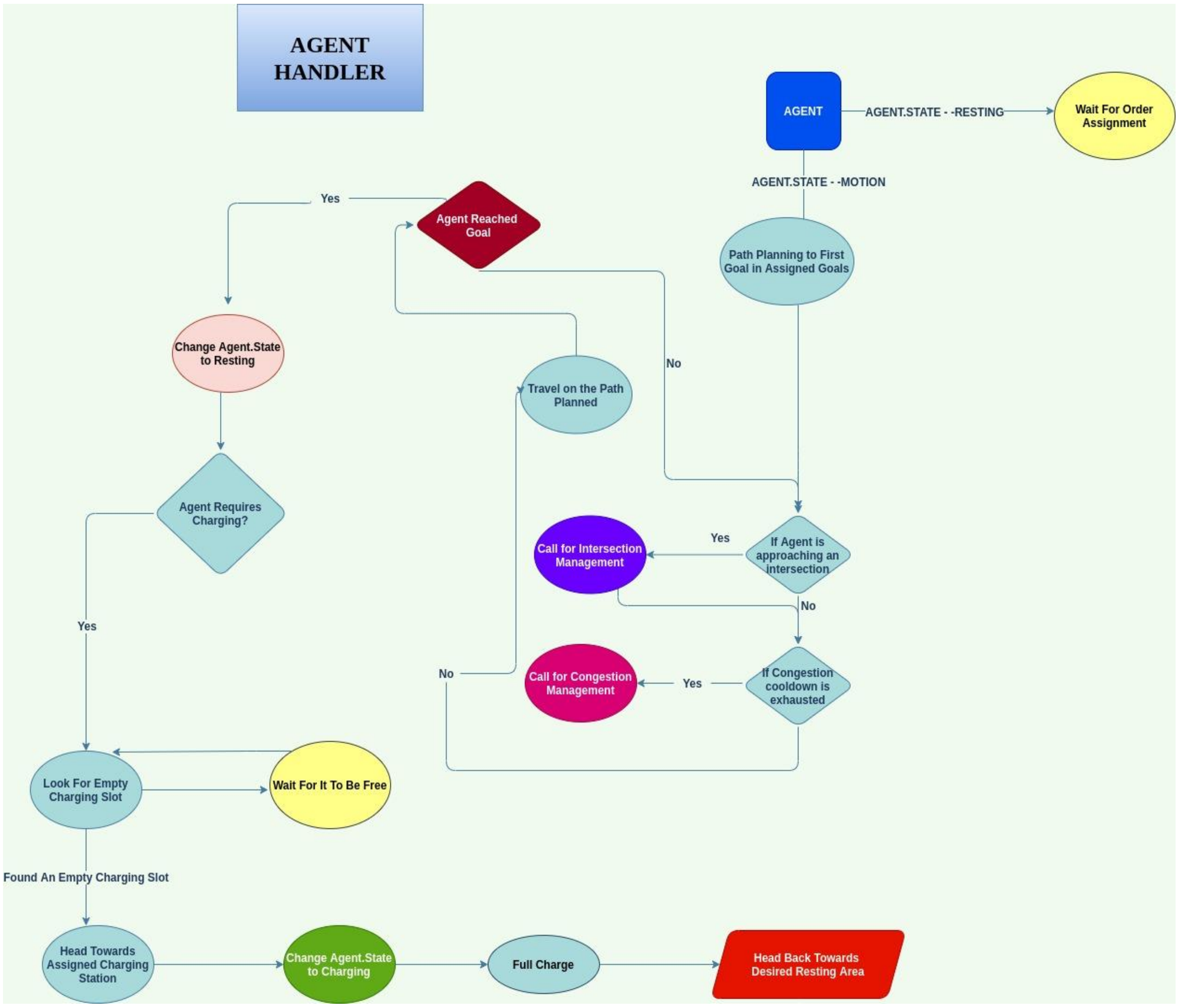


```
def update_every_agent():
    for road in Roads:
      for agent in road:
        lead-> next_agent in road after agent # if last then lead->None
        if lead==None: # meaning last agent in road
          passing="Not Allowed"
          if distance_to_intersection<threshold_to_intersection:
              if booking(Intersetion):
                  passing="Allowed" # Allowed to pass
          if passing=="Not Allowed":
              agent.stop()
              continue
          if distance_to_intersection less than interesection range->
            Remove(Road,agent)
            intersection.agent=agent

        update(agent,lead,dt) # dt-> control time
    for intersection in all_intersection:
        # Check if agent in it can move in direction specifiec
        if intersection.agent can go:
            Append(New_Road,agent)
            Remove(inte)
```

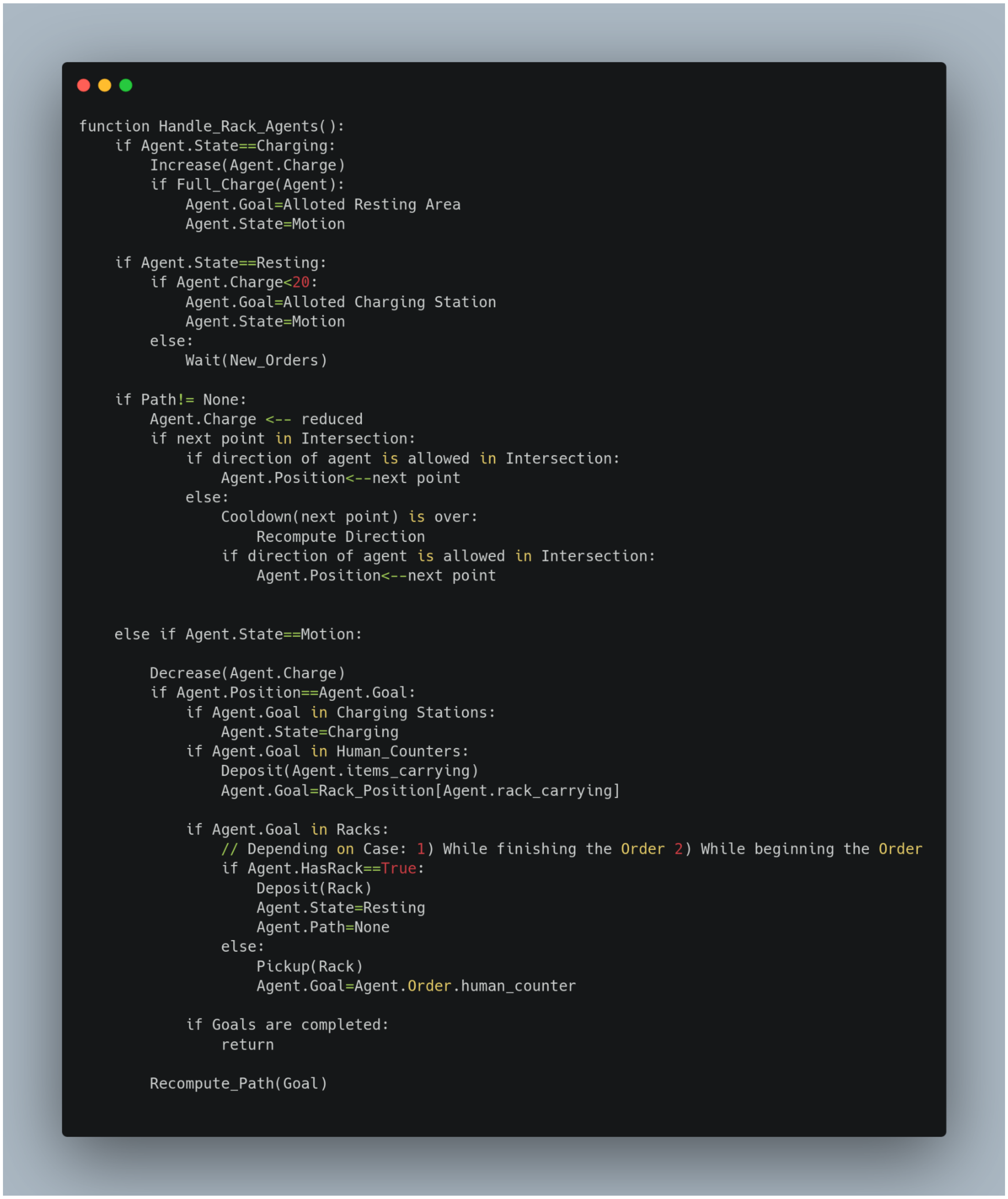


```
# In accordance with IDM
def update(self, lead, dt): # self-> current bot,  lead-> bot ahead of it on road
        # Update position and velocity
        # self.a_max-> maximum accleration
        # slef.b_max-> comfortable deacceleration
        if self.v + self.a*dt < 0:
            self.x -= 1/2*self.v*self.v/self.a
            self.v = 0
        else:
            self.v += self.a*dt
            self.x += self.v*dt + self.a*dt*dt/2

        # Update acceleration
        alpha = 0
        if lead:
            delta_x = lead.x - self.x - lead.l
            delta_v = self.v - lead.v
            self.sqrt_ab = 2*np.sqrt(self.a_max*self.b_max)
            alpha = (self.s0 + max(0, self.T*self.v + delta_v*self.v/self.sqrt_ab)) / delta_x

        self.a = self.a_max * (1-(self.v/self.v_max)**4 - alpha**2)

        if self.stopped:
            self.a = -self.b_max*self.v/self.v_max
```

8.4 Block Diagram for Agent Handler

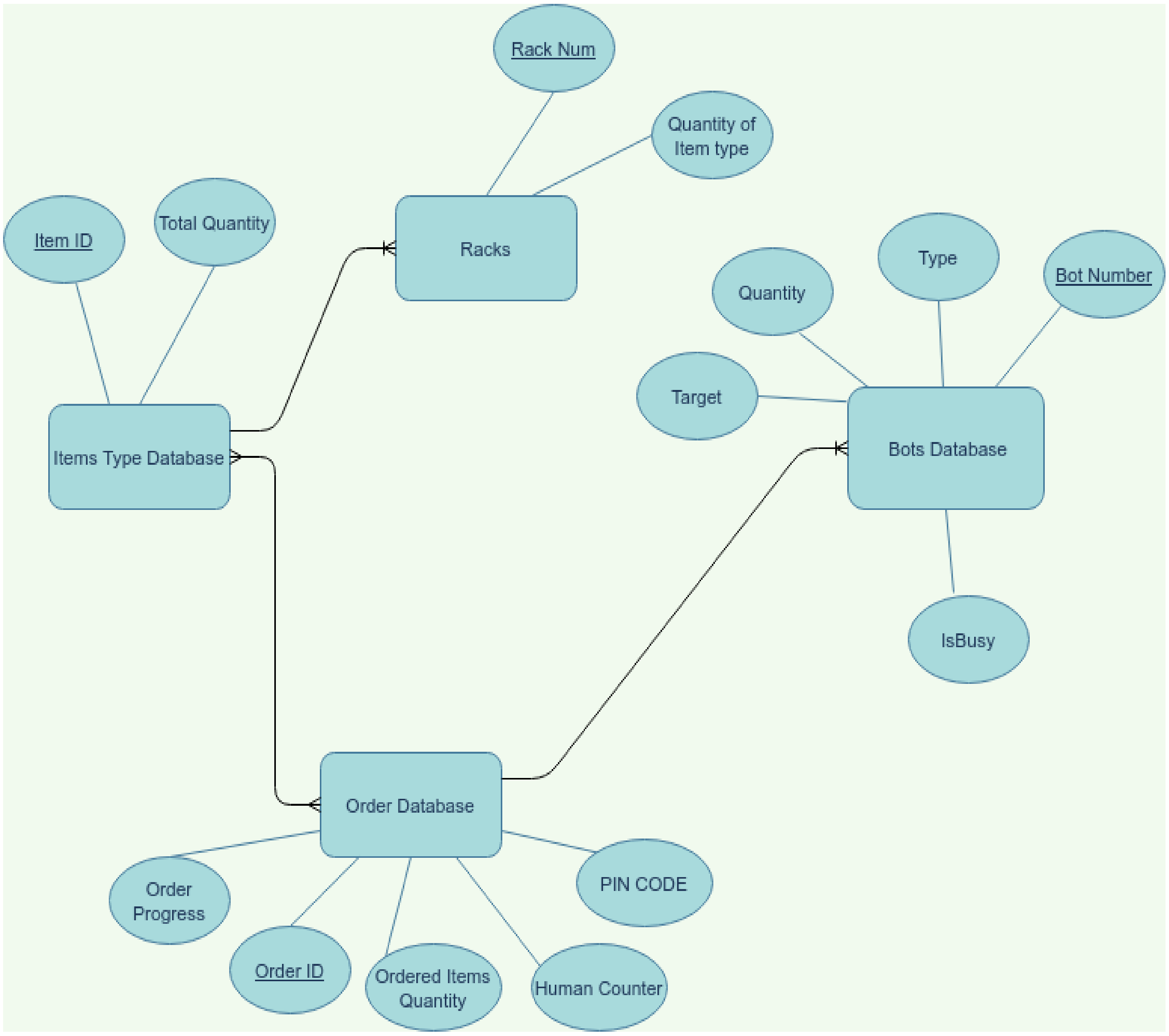

AGENT HANDLER
AGENT
AGENT.STATE - -RESTING
Wait For Order Assignment
AGENT.STATE - -MOTION
Path Planning to First Goal in Assigned Goals
Agent Reached Goal
Yes
No
Travel on the Path Planned
Change Agent.State to Resting
Agent Requires Charging?
Yes
If Agent is approaching an intersection
Yes
Call for Intersection Management
No
If Congestion cooldown is exhausted
Yes
Call for Congestion Management
No
Look For Empty Charging Slot
Wait For It To Be Free
Found An Empty Charging Slot
Head Towards Assigned Charging Station
Change Agent.State to Charging
Full Charge
Head Back Towards Desired Resting Area

8.5 Pseudo Code for Agent Handler



```
function Handle_Rack_Agents():
    if Agent.State==Charging:
        Increase(Agent.Charge)
        if Full_Charge(Agent):
            Agent.Goal=Alloted Resting Area
            Agent.State=Motion

    if Agent.State==Resting:
        if Agent.Charge<20:
            Agent.Goal=Alloted Charging Station
            Agent.State=Motion
        else:
            Wait(New_Orders)

    if Path!= None:
        Agent.Charge <-- reduced
        if next point in Intersection:
            if direction of agent is allowed in Intersection:
                Agent.Position<--next point
            else:
                Cooldown(next point) is over:
                    Recompute Direction
                if direction of agent is allowed in Intersection:
                    Agent.Position<--next point


    else if Agent.State==Motion:

        Decrease(Agent.Charge)
        if Agent.Position==Agent.Goal:
            if Agent.Goal in Charging Stations:
                Agent.State=Charging
            if Agent.Goal in Human_Counters:
                Deposit(Agent.items_carrying)
                Agent.Goal=Rack_Position[Agent.rack_carrying]

            if Agent.Goal in Racks:
                // Depending on Case: 1) While finishing the Order 2) While beginning the Order
                if Agent.HasRack==True:
                    Deposit(Rack)
                    Agent.State=Resting
                    Agent.Path=None
                else:
                    Pickup(Rack)
                    Agent.Goal=Agent.Order.human_counter

            if Goals are completed:
                return

        Recompute_Path(Goal)
```

9. ER Diagram

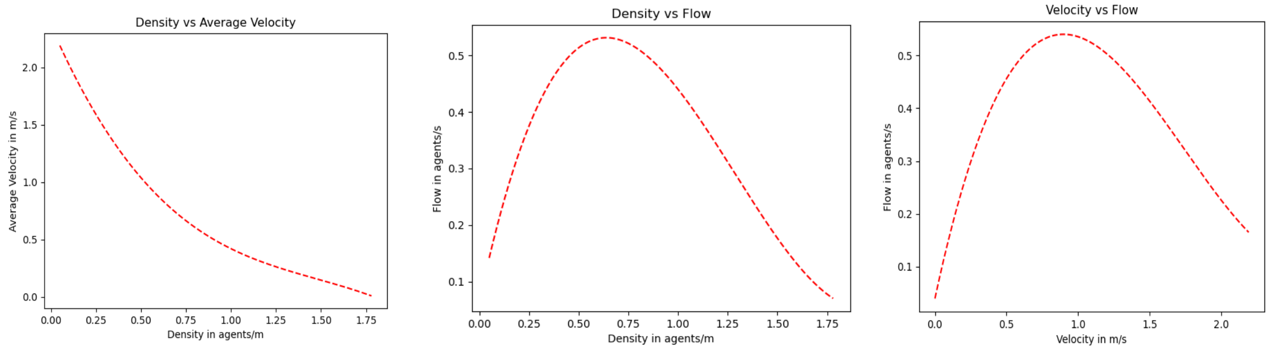


Rack Num
Quantity of Item type
Racks
Item ID
Total Quantity
Items Type Database
Quantity
Type
Bot Number
Target
Bots Database
IsBusy
Order Database
Order Progress
Order ID
Ordered Items Quantity
Human Counter
PIN CODE

10. Results

The results were simulated for different scenarios under different settings of density and number of orders. A sample simulation run is given as a video in the supplementary material. After the simulation, we first analysed the performance of the network using fundamental diagrams. The data was collected from a large number of simulations, and the plots were smoothened by using curve fitting. The three fundamental diagrams for our warehouse simulation are given in Fig. 2. Fig. 2(a) shows the reduction in speed along with the increasing density as a response to the agents not getting enough space. Fig. 2(b) shows an initial increase in flow as more robots use the road, while the flow reduces after the capacitated flow due to congestion in which case the robots move at significantly smaller speeds. Similarly, Fig. 2(c) shows smaller speeds giving smaller flow due to congestion, while larger speeds give a smaller flow due to under-utilisation of the road.

Fig. 7 compares observed vs predicted density averaged over all the agents in the warehouse which is what we use in strategy 2 for a better prediction of trajectory. The figure shows how precisely the agents predict the traffic density.



(a) (b) (c)

Figure 2. Fundamental Diagrams (a) Density vs Avg Velocity, (b) Density vs Flow, (c) Velocity vs Flow

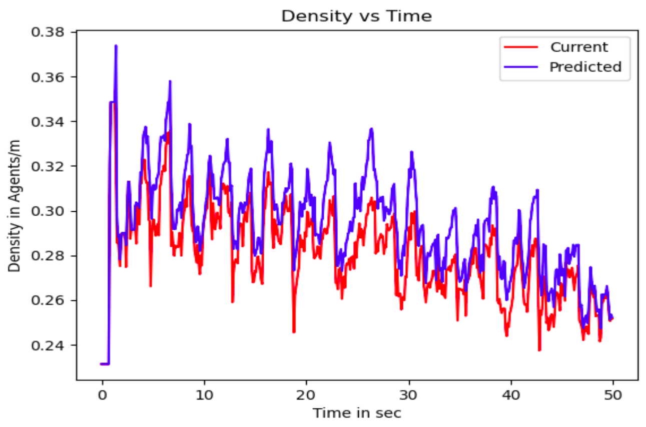


Figure 3. Comparison between Current and Predicted Density

The congestion management system is further tested. We compare the two strategies against a system where the robots use shortest path algorithms for routing, neglecting densities. The performance is tested with the help of two metrics that are time to fulfil all orders and number of item picks per 100 seconds.

TABLE II. PERFORMANCE USING THE TIME TO FULFIL METRIC

#	Orders	Density	Time to Fulfil			Picks per 100 secs		
			St 0	St 1	St 2	St 0	St 1	St 2
1	40	.241	4374	**4334**	**4334**	9.14	**9.23**	**9.23**
2	40	.245	3774	3814	**3774**	10.60	10.49	**10.60**
3	60	.256	3774	3774	**3734**	15.90	15.90	**16.07**
4	80	.259	4081	**3934**	3994	19.60	**20.34**	20.03
5	120	.273	7934	7293	**6734**	15.12	16.45	**17.82**
6	80	.289	4813	4773	**4573**	16.62	16.76	**17.50**
7	100	.294	5374	5354	**5314**	18.61	18.68	**18.82**
8	120	.310	5854	5854	**5694**	20.50	20.50	**21.07**
9	160	.314	6115	5854	**5734**	26.17	27.33	**27.90**
10	200	.350	7174	6954	**6874**	27.88	28.76	**29.10**

St 0: No congestion control, St 1: Strategy 1 (static), St 2 Strategy 2 (anticipatory)

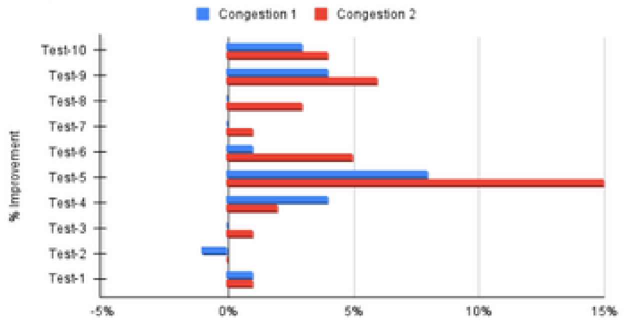


Fig 4 . Percentage improvement w.r.t. planning without congestion.

To do a better comparison between the two strategies, we did testing for very different scenarios like low density, moderate density and high density. Table II shows the comparison table between results using strategy 1 (static) and strategy 2 (anticipatory), where each test case is in an increasing order of average density. Fig. 4 specifically shows the percentage improvement as compared to planning without congestion control. It can be seen that inclusion of congestion control reduces the time of fulfilment appreciably, while the reduction is much better for the anticipatory strategy. We did 10 tests with different densities. We ran simulations in all the test cases for 1000 timesteps of our simulator.

Congestion 2 showed a great performance over various testcases , especially on the high density testcases. However, when we tried probabilistic modelling of density estimation for the bots we found it to be performing better than the Congestion 2. Fig 5. Shows the % Improvement over the original Congestion 2 by using the Congestion 2 (Extended).

		Time to Fulfil	
#	**Density**	**C-2**	**C-2(Extended)**
1	0.281	8294	**8034**
2	0.295	3282	**3214**
3	0.31	7081	**6774**
4	0.327	7333	**6634**
5	0.334	8395	**7214**

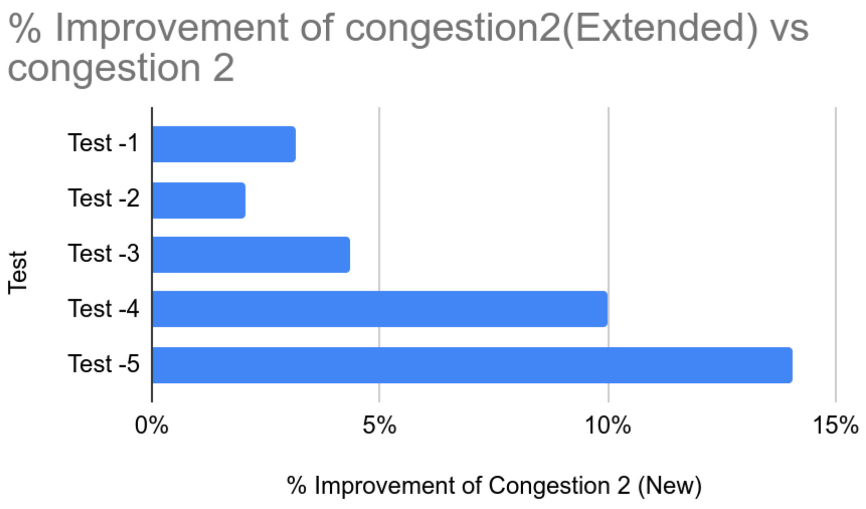


Fig 5. Percentage improvement of congestion 2(Extended) vs congestion 2

An example from our simulation for how a bot re-evaluates its path according to congestion scenarios around it using scheme 1 is shown in Fig. 5. The figure displays a time-wise comparison of the path for a single bot that is adjusting its path whenever it reaches an intersection by finding a path through low congestion.

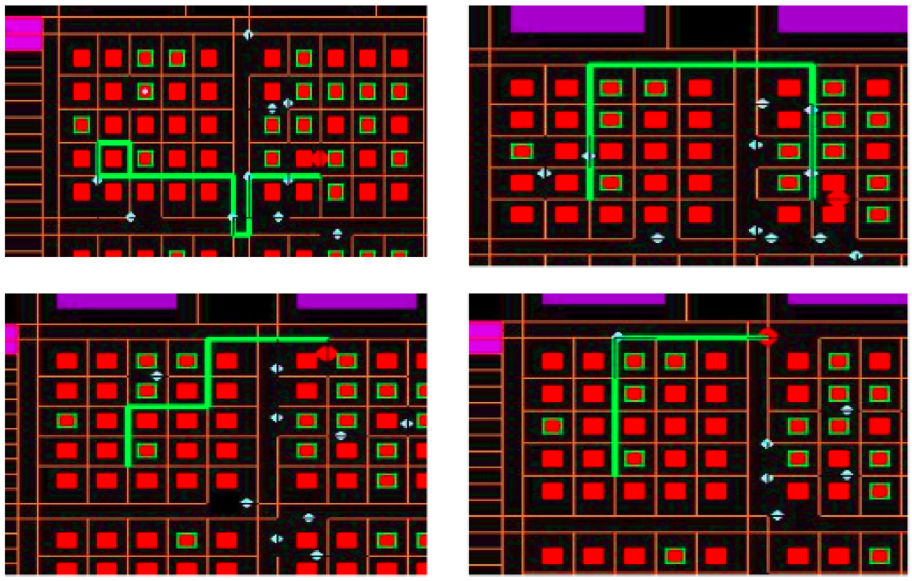


Figure 6. Simulation Result for Congestion Scheme -1 showing agent re-planning and getting a better path.

11. Conclusion

The number of robots in a warehouse are rapidly increasing that pose new problems worthy of research. In this paper we develop a simulator that can be used for research over the problems of routing, intersection management, congestion management, picking order, etc., the results of which will be based on interactions of all warehousing components rather than being based on avoidable assumptions. Due to the sudden surge in the demand for warehouses and the following increase in the robot density, it is now important to critically analyze the transportation network within the warehouses against the characteristics that were done in the paper using the fundamental diagrams. Congestion is one of the major reasons for bottlenecks, which this paper eradicated by estimating the density benefitting from a fully automated nature of the warehouse and making the robots select routes that maximize expected speeds. Simulations show that the anticipatory congestion avoidance performs better than the system that assumes a static density, replanning often, and a system that routes the robots without considering the congestions. Future efforts will be on benchmarking the simulator against ground truths taken from real warehouses, partnering with industry for real order databases, and improving the network design using the analysis reported in the paper.

12. References

[1] L. Nicolas, F. Yannick, H. Ramzi, "Order batching in an automated warehouse with several vertical lift modules: Optimization and experiments with real data." *European Journal of Operational Research*, vol. 267, no. 3, pp. 958-976, 2018.

[2] H. Yoshitake, R. Kamoshida and Y. Nagashima, "New Automated Guided Vehicle System Using Real-Time Holonic Scheduling for Warehouse Picking," in *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1045-1052, April 2019.

[3] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor and P. J. Stuckey, "Integrated Task Assignment and Path Planning for Capacitated Multi-Agent Pickup and Delivery," in *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816-5823, July 2021..

[4] Cyberbotics Ltd., “Webots: Professional Mobile Robot Simulation” Available at: https://cyberbotics.com/

[5] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 2149-2154 vol.3.

[6] E. Rohmer, S. P. N. Singh and M. Freese, "V-REP: A versatile and scalable robot simulation framework," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321-1326.

[7] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” Available at: https://github.com/bulletphysics/bullet3/blob/master/docs/pybullet quickstartguide.pdf, Tech. Rep.

[8] C. Pinciroli, V. Trianni, R. O’Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, M. Dorigo, “ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems,” *Swarm Intelligence*, volume 6, number 4, pages 271-295. Springer, Berlin, Germany, 2012.

[9] F. Christianos, L. Schäfer, S. Albrecht. "Shared experience actor-critic for multi-agent reinforcement learning." *Advances in Neural Information Processing Systems*, vol. 33, pp. 10707-10717, 2020.

[10] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575-2582.

[11]A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, “CARLA: An open urban driving simulator,”. In *Conference on robot learning*, pp. 1-16, PMLR, 2017.

[12] C. Street, S. Pütz, M. Mühlig, N. Hawes and B. Lacerda, "Congestion-Aware Policy Synthesis for Multirobot Systems," in *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 262-280, Feb. 2022.

[13] V. Digani, L. Sabattini and C. Secchi, "A Probabilistic Eulerian Traffic Model for the Coordination of Multiple AGVs in Automatic Warehouses," in *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 26-32, Jan. 2016.

[14] A. E. Abdelaal, M. Sakr and R. Vaughan, "LOST Highway: A Multiple-Lane Ant-Trail Algorithm to Reduce Congestion in Large-Population Multi-robot Systems," *2017 14th Conference on Computer and Robot Vision (CRV),* 2017, pp. 161-167.

[15] L. S. Marcolino and L. Chaimowicz, "Traffic control for a swarm of robots: Avoiding target congestion," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 1955-1961.

[16] K. Srivilas and P. Cherntanomwong, "Routing algorithm in warehouse with congestion consideration using an ACO with VLC support," *2017 International Electrical Engineering Congress (iEECON)*, 2017, pp. 1-4.

[17] K. Lerman and A. Galstyan, “Mathematical model of foraging in a group of robots: Effect of interference,” *Autonomous Robots*, vol. 13, no. 2, pp. 127–141, 2002.

[18] Y. T. dos Passos, X. Duquesne, L. S. Marcolino. "Congestion control algorithms for robotic swarms with a common target based on the throughput of the target area." *arXiv e-prints* (2022): arXiv-2201.

[19] M. Treiber, A. Hennecke, D. Helbing. "Congested traffic states in empirical observations and microscopic simulations." *Physical review E*, vol. 62, no. 2, 1805, 2000.