Graph-Based Behavior Modulation for Enhancing Multi-Agent Navigation*

Javier Perera-Lago[†1], Francisco-Jose Campos-Castro[‡1], Jérôme Guzzi[§2], and Rocio Gonzalez-Diaz[¶1]

[1]Departamento de Matemática Aplicada I, Universidad de Sevilla, Seville, Spain
[2]SUPSI, IDSIA, Lugano, Switzerland

Abstract

Autonomous navigation in constrained environments often leads to deadlocks and collisions, especially in narrow or crowded spaces. We present a graph-based behavior modulation integrated into the Navground simulator, a playground to experiment with navigation algorithms, to improve the performance of standard navigation strategies. By modeling proximity relationships between agents and obstacles, the modulation detects potential blockages and adapts agents' behavior to prevent them through actions such as yielding or rerouting. Experiments show a significant reduction in deadlocks and collisions, demonstrating the effectiveness of this graph-based approach.

1 Introduction

Recent studies have explored advanced computational techniques for modeling fleet behavior in dynamic environments. For instance, [1] developed a data-driven approach using deep reinforcement learning to optimize fleet routing and mitigate collision risks in urban settings. The discrete modeling of fleet behavior presents challenges in ensuring reliability and safety. Our objective is to improve the safety and efficiency of autonomous navigation by developing a behavior modulation that models the agents' environment by abstracting their geometric characteristics and focusing solely on the proximity relationships between them and other objects to prevent collisions and blockages in narrow or crowded environments.

2 Navground Navigation Simulator

Hinted by its name, Navground[2] is a playground to experiment with navigation algorithms. At its core, the simulator operates with multi-agent systems that carry out specific navigation tasks. An agent A is represented as a tuple (p, r, α, m), being $p = (p_x, p_y)$ its center, r its radius, α an orientation indicating the direction of movement and m a safety margin to control whether the agent is in danger of a collision, as shown in Fig. 1. Agent A can detect walls, obstacles, and other agents (called neighbors) within its horizon length of 2ℓ. Each agent executes a local navigation algorithm (called a *behavior*) that, without communicating with other agents, reactively computes control commands based on the positions and velocities of nearby obstacles and neighbors.

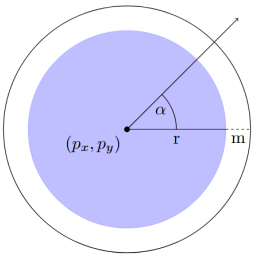

Figure 1: An agent with center $p = (p_x, p_y)$, radius r, orientation α and safety margin m.

Navground offers the possibility to create custom scenarios where to run the simulations. In this work we have considered the following three to test the navigation behavior and the proposed graph-based modulation (see Fig. 2): a) *Bowtie scenario*, which represents a 30m long and 3m wide corridor with both ends connected, with a narrow section of 1.5m width in the center. This scenario is initialized with 4 agents, 2 navigating to the right, and 2 navigating to the left. It is used to verify that when the graph-based modulation is not enabled, the agents have a frontal collision in the narrow region, but they do not when the modulation is enabled. b) *Crowded Corridor Scenario*, which represents a 30m long and 3m wide corridor with both ends connected. It is initialized with 5 agents, 3 navigating in parallel to the right, and 2 navigating in parallel to the left. Both groups of agents block the corridor. This scenario is used to check that when the modulation is not enabled, the agents do not solve the blockage, causing a frontal collision in the middle of the corridor, while they do unblock the corridor when the graph-based modulation

*Research supported by REXASI-PRO H-EU project, Grant agreement ID: 101070028, Acción V.1A2 del VII Plan de Investigación y Transferencia, Anualidad 2025,
[†]Email: jperera@us.es
[‡]Email: fracamcas1@alum.us.es
[§]Email: jerome.guzzi@idsia.ch
[¶]Email: rogodi@us.es

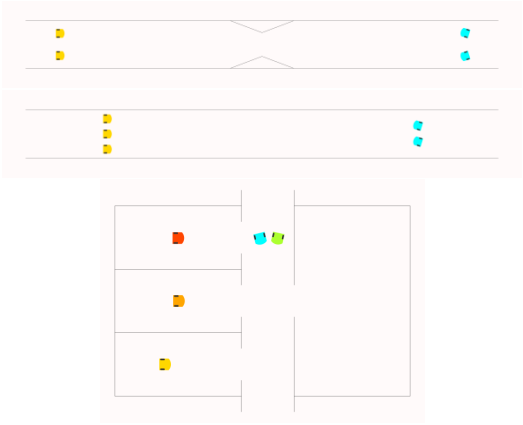

Figure 2: Top: Bowtie Scenario. Middle: Crowded Corridor. Bottom: Home Scenario.

is enabled. c) *Home scenario*, which represents 3 areas simulating individual rooms, each 3m wide and 6m long. On the right, there is a rectangle simulating a 5.5m wide and 9m long common area. Between them, there is a 2.5m wide corridor that can be blocked by two agents navigating in parallel. The 3 individual rooms and the common area are connected to the central corridor through 1.5m wide doors, creating four local narrow regions. This scenario is initialized with 5 agents where 3 are assigned the task of continuously traveling back and forth between their individual room and the common area, crossing the corridor. The remaining 2 continuously navigate up and down the corridor. This scenario is used to verify that the graph-based modulation significantly improves the navigation behavior in a complex environment.

3 Problem to solve

We aim to avoid two negative events: a) *deadlocks*, that occur when agents cannot move due to the positions of other agents; b) *collisions*, when an agent collides with other agents or with static obstacles.

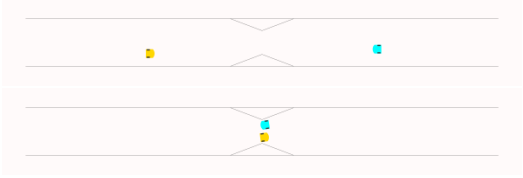

Figure 3: A narrow door (top), where agents navigating in opposite directions block each other (bottom).

See for example the scenario depicted in Fig. 3. The agent on the top-left aims to move to the right through the corridor, while the agent on the top-right goes to the left. Their optimal speeds are identical, and they are at the same distance from the central door, so both will reach the door at the same time but from opposite directions. The door is not wide enough to allow two agents to pass simultaneously, so they block each other.

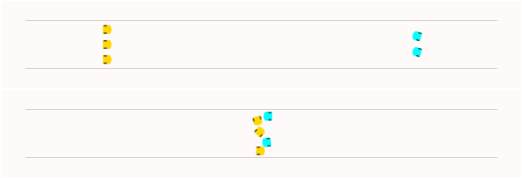

Figure 4: Crowded corridor scenario with agents navigating in parallel (top), creating a blockage (bottom).

A similar situation occurs in the scenario depicted in Fig. 4. On the top-left, three agents go in parallel to the right, while on the top-right, two agents go in parallel to the left. The agents do not detect that this formation is obstructive, and when the agents meet head-on in the middle of the corridor, they block each other.

To prevent and avoid such potential risks, we propose an algorithm to modulate any kind of navigation behavior that the agents may be using.

4 Proposed solution

Each agent, at each time step, gathers information about its immediate environment such as the position, velocity, and size of its nearest neighbors, as well as the position of fixed obstacles like walls, and executes a command to achieve its goal (reaching a target point or following a direction) after an optimization process that depends on its behavior and the kinematic characteristics of the agent. The proposed graph-based modulation modifies the agent's behavior in two distinct ways: a) *a priori*, modifying the agent's perception of itself and its environment (by creating virtual obstacles that do not exist in the real environment) or its motion parameters (such as optimal speed or relaxation time) before the motion command is calculated; b) *a posteriori*, modifying the motion command after it has been calculated (by changing its direction or setting it to zero to make the agent stop).

4.1 Methodology

First, we divide the agent's neighbors into 3 categories: a) *stuck neighbors*, that move at a speed lower than 0.01ms^{-1}; b) *same-flow neighbors,* that move at a speed greater than 0.01ms^{-1} and their orientations differ from A's by at most 60^o; c) *opposite-flow neighbors* that move at a speed greater than 0.01ms^{-1} and their orientations differ from A's by more than 60^o. The graph-based modulation treats stuck neighbors like static obstacles. Opposite-flow neighbors help manage narrow passages, while same-flow neighbors help to check if A is in a blocking platoon.

After splitting A's neighbors into these three categories, the next step is to build the so-called F-graph $\mathrm{F} = (V_\mathrm{F}, E_\mathrm{F})$, where: V_F is the set of nodes corresponding with fixed obstacles (walls, static obstacles and stuck neighbors); given two obstacles $v_1, v_2 \in V_\mathrm{F}$, the pair $\{v_1, v_2\}$ is an edge of E_F if $d(v_1, v_2) < 2r+2m$, which is the minimum space needed for agent A to pass between them. In particular, adjacent walls are nodes of V_F connected by an edge of E_F. See Fig. 5.

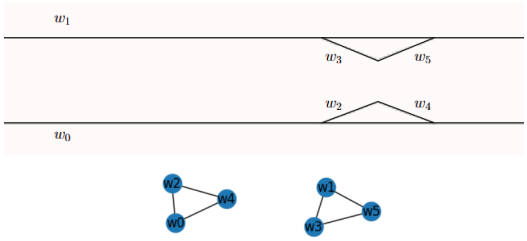

Figure 5: Top: An example of a scenario with six walls as fixed obstacles. Bottom: F-graph of the scenario.

We define the set of *narrow pairs* as:

$$N = \{\{v_1, v_2\} \subset V_\mathrm{F} \mid 2r+2m < d(v_1, v_2) \leq 4r+3m\},$$

that includes pairs of obstacles that let one, but not two, agents pass at once. In Fig. 5, the narrow pairs are $N = \big\{\{w_2, w_3\}, \{w_2, w_5\}, \{w_4, w_3\}, \{w_4, w_5\}\big\}$.

Figure 6: Narrow region of the scenario from Fig. 5.

The *narrow region* of $\{v_1, v_2\} \in N$ is defined as:

$$R(\{v_1, v_2\}) = \big\{p \in \mathbb{R}^2 \mid d(p, f_1), d(p, f_2) < 3r + 2m\big\},$$

and includes the points such that, if an agent's center is located there, it blocks passage for its neighbors. The *global narrow region* is defined as:

$$R = \bigcup_{\{v_1, v_2\} \in N} R(\{v_1, v_2\}).$$

Each connected component of R is called a *local narrow region*. Fig. 6 pictures a global narrow region.

Agent A checks for interaction with nearby narrow regions by projecting a segment of length ℓ (half of its horizon) from (p_x, p_y) in direction α. If it intersects narrow regions, the closest region, R_min, is considered.

Then if $(p_x, p_y) \in R_\mathrm{min}$, agent A is already blocking and must move forward to exit R_min. However, if the projection intersects R_min but $(p_x, p_y) \notin R_\mathrm{min}$, agent A must check if there are other opposite-flow neighbors near R_min: a) If there is no opposite-flow neighbor in front of A or none of them is approaching R_min, agent A does not need to modulate its behavior; b) if there is an opposite-flow neighbor within R_min, agent A must stop immediately by setting its optimal speed to 0; c) if there is an opposite-flow neighbor but not within R_min, agent A must decide whether to wait or continue while the other agent waits. To make this decision, A uses the trajectory projections and current velocities of both itself and its opposite-flow neighbor to estimate which agent will reach R_min first. If A is expected to arrive first, it proceeds; otherwise, it stops until the opposite-flow neighbor clears R_min.

Same-flow neighbors are not considered in this part of the procedure because they move in a similar direction to that of A.

If agent A concludes to not stop in front of a local narrow region, it has to check if it is part of a platoon of agents that is blocking a non-narrow region. The so-called SA-graph $\mathrm{SA} = (V_\mathrm{SA}, E_\mathrm{SA})$ is built, where: V_SA is the set formed by the agent A and its same-flow neighbors; given $v_1, v_2 \in V_\mathrm{SA}$, the pair $\{v_1, v_2\}$ belongs to E_SA if $d(a_1, a_2) < 2r + 2m$. This graph shows the proximity relationships between agent A and its same-flow neighbors. Here, an edge between two nodes indicates that the corresponding agents are so close that A cannot pass between them.

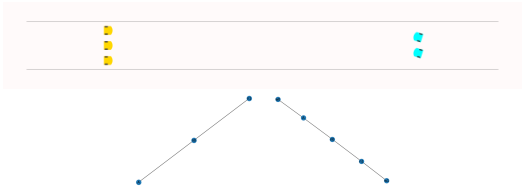

Figure 7: Top: a corridor scenario blocked by the yellow agents where A is one of them. Bottom-left (resp. -right): PA (resp. FPA) graph for agent A.

Next, we consider the PA-graph called A's platoon, which is the subgraph of the SA-graph generated by the path-connected component of A. The next step is to build the so-called FPA-graph $\mathrm{FPA} = (V_\mathrm{FPA}, E_\mathrm{FPA})$ where: $V_\mathrm{FPA} = V_\mathrm{F} \cup V_\mathrm{PA}$; $E_\mathrm{FPA} = E_\mathrm{F} \cup E_\mathrm{PA} \cup \big\{\{v_1, v_2\}$ with $v \in V_\mathrm{F}, w \in V_\mathrm{PA}$ and $d(v_1, v_2) < 2r + 2m\big\}$. Fig. 7 shows an example of the PA and FPA graphs for an agent A in the Crowded Corridor Scenario.

Knowing the F, PA, and FPA graphs, we can decide whether A's platoon is blocking. Let C be the set of vertices of V_F that are adjacent in FPA to any vertex of PA, and let F_C be the subgraph of F generated by the vertices of C. In other words, F_C is the graph formed by the fixed obstacles that are so close to A's platoon that it is not possible to pass between them and the platoon. If C is empty, it means that A's platoon is not near any fixed obstacles and therefore it is not blocking. If C is not empty, and the graph F_C is path-connected, this means that A's platoon is near a set of consecutive fixed obstacles, and then A's platoon is not blocking. However, if the graph F_C is not path-

connected, this means that there exists a path in FPA that starts at a fixed obstacle v_1, passes through one or more agents of the platoon, and ends at another fixed obstacle v_2 that is not near v_1. That is, the platoon has created a physical barrier connecting two fixed obstacles that were initially sufficiently separated to allow the free movement of other agents in the opposite direction. This situation is called a *blockage*. For example, consider the scenario from Fig. 7 where $C = \{w_0, w_1\}$ being w_1 and w_2 the two walls that are adjacent in the FPA-graph to some vertex from the PA-graph. In the F-graph, the vertices w_0 and w_1 are not path-connected because their respective walls are far enough. This means A and its neighbors form a blockage spanning from one wall to the other.

When agent A detects that it is part of a platoon that is creating a blockage, it acts as follows: first, agent A counts how many neighbors from its platoon are ahead of it. Suppose it counts n agents ahead. In that case, it multiplies its optimal speed by $(3/4)^n$, causing the agents at the front of the platoon to continue at their usual pace while the agents behind slow down to create space; second, agent A check how many neighbors from its platoon are to its right. If there are none, it does nothing. If there is at least one, it makes the navigation behavior detect a new virtual wall having one endpoint in front of A and the other endpoint to its left. The navigation behavior, seeing this wall on its left side, will cause the agent A to move to the right, freeing up space on the left. The agent will modulate its behavior until it no longer detects that its platoon is creating a blockage.

After this *a priori* modulation ends, the behavior computes the motion. The *a posteriori* modulation restores optimal speed and clears virtual walls.

4.2 Experiments

We ran 200 simulations of 120 s each, with all agents executing the *Human-Like (HL)* behavior [3], a computationally light algorithm inspired by pedestrian motion, which follows three steps:1) it selects the best direction toward the target while avoiding potential collisions by considering a safety margin around the agent and the velocity of nearby entities; 2) it determines an appropriate speed that allows the agent to stop within a safe distance if needed; 3) the velocity is smoothly adjusted over time to ensure natural movement transitions. In 100 simulations, the agents navigate using HL without modulation, while in the other 100, the agents navigate using the modulation. Experiments with other behaviors are explored in the Appendix. We introduce randomness into the initialization of the 200 simulations. The 3 agents within individual rooms always start the simulation in their respective rooms, but in a random position and orientation, determined by the seed. Meanwhile, the 2 agents navigating the corridor always start the simulation facing downward, but in a different position within the corridor, also dependent on the seed. After running each simulation, we counted how many collisions occurred. We slightly adapted the *navground.sim* code to better reflect real count of deadlocks and collisions. A summary of the number of collisions and deadlocks per simulation can be seen in Fig. 8.

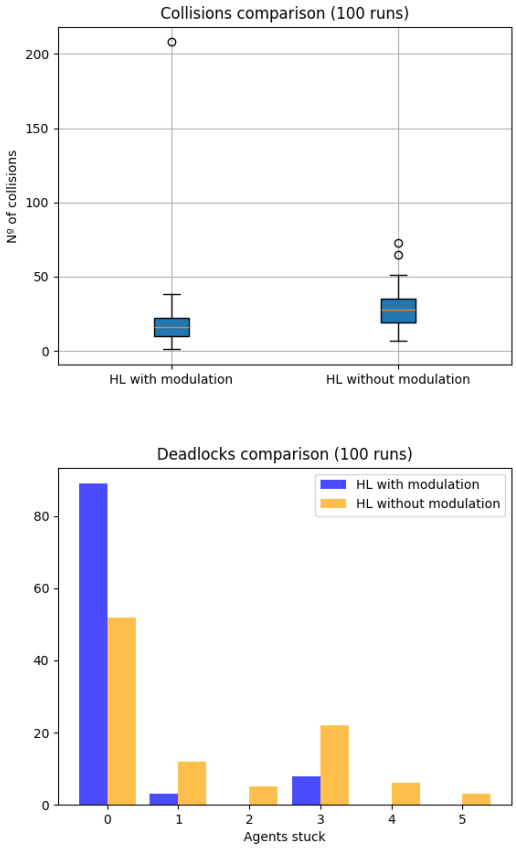

Figure 8: Collisions and deadlocks between HL behavior with and without graph-based modulation.

Code availability: `https://github.com/Cimagroup/navground_graphs`.

References

[1] W Zhang, Q Wang, J Li, et al. "Dynamic fleet management with rewriting deep reinforcement learning". In: *IEEE Access* (2020). DOI: doi:10.1109/ACCESS.2020.301407.

[2] IDSIA Robotics Lab. *Navground 0.5.0 Documentation*. 2025. URL: https://idsia-robotics.github.io/navground/0.5/index.html.

[3] J Guzzi, A Giusti, LM Gambardella, et al. "Human-friendly robot navigation in dynamic environments". In: *IEEE ICRA 2013*. 2013. DOI: 10.1109/ICRA.2013.6630610.

A The Modifications made on Navground.sim

Collisions can either be between two agents or between an agent and a wall. This count can be done using the functions provided by the *navground.sim* module. We tracked how many of the 5 agents got stuck during the simulation, meaning they couldn't move until the end. To do this, we slightly adapted the *navground.sim* code to better reflect real count of deadlocks and collisions.

The *navground.sim* considers an agent to be stuck if it has not moved from a specific point in time until the end of the simulation. However, an agent may stop voluntarily because it is near a narrow area and is waiting for another agent to pass. If the simulation ends right at that moment, the system might interpret that the agent is permanently stuck, when in fact it is only temporarily waiting. For this reason, we only count as real deadlocks those in which agents did not stop voluntarily but because they were truly unable to move. Finally, we also register the efficacy of the agents along the simulation. The efficacy of an agent at a given moment is the ratio between its actual speed and its target speed. It is a value between 0 and 1, with higher values being better, the closer it is to 1. The value we will calculate is the average efficacy across all agents and all simulation steps. We check how much the HL behavior improves when the graph-based modulation is applied. The median number of collisions per simulation decreases from 28 to 16 when the modulation is applied, indicating a significant improvement in the safety of the agents and their users. A broader summary of the number of collisions and deadlocks per simulation can be seen in Fig. 8. It can also be observed that the probability of an agent getting stuck decreases significantly. When the modulation is not applied, only 50% of the simulations end without agents in a deadlock, while this percentage rises to more than 80% when the modulation is applied. Fig. 8 shows details on the number of deadlocks in the simulations. Finally, we check that the average efficacy also improves slightly when the graph-based modulation is applied to the HL behavior. When the behavior is not modulated, the average effectiveness is 24.95%, while when the modulation is applied, the average effectiveness rises to 25.22%. It can be concluded that the performance of the HL behavior in the Home scenario improves in all the studied aspects when the graph-based modulation is applied.

B Malta Cross Scenario

We have included in the code available another custom scenario: *Malta cross scenario*, which represents an intersection between two corridors, one horizontal and one vertical. Both corridors are connected at their ends. The shape of the corridors is intentionally unrealistic so that the intersection forms a narrow region that allows only one agent to pass at a time. This scenario is initialized with 4 agents: one moving left in the horizontal corridor, one moving right in the horizontal corridor, one moving up in the vertical corridor, and one moving down in the vertical corridor. It is used to verify that when the modulation is enabled, the agents also detect other agents moving in perpendicular directions. see fig 9.

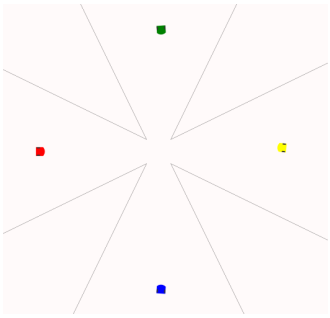

Figure 9: Initialization of the Malta cross scenario.

C ORCA and SF Behaviors

Other standard behaviors available at the Navgraound platform are: a) the *Optimal Reciprocal Collision Avoidance (ORCA)* behavior designed to avoid collisions cooperatively and efficiently by adjusting their velocities based on the positions and velocities of other agents in their immediate environment; b) the *Social Force Model (SFM)* behavior describes pedestrian dynamics using a framework based on fictitious forces: a pedestrian's movement is driven by a force that directs them towards their desired goal, while other forces repel them from obstacles and other pedestrians, capturing the behavior and social interactions of pedestrians in crowded environments.

C.1 Experiments with the ORCA and SF Behavior on the Home Scenario

We repeated the analysis for the simulations using ORCA as the baseline navigation behavior. The median number of collisions per simulation decreases from 6 to 0 when the modulation is applied. In fact, Fig. 10 shows that 75% of the simulations that apply ORCA with the graph-based modulation only have at most one collision during their 120 seconds. With respect to the number of deadlocks, the results slightly worsen when the modulation is applied. When no modulation is applied, 100% of the simulations end without deadlocks, while when the modulation is applied, we have two simulations with one agent blocked and one with two agents blocked. Anyway, 97% of the simulations end with all agents navigating freely, which is still a good result. The average efficacy decreases from 7.38%

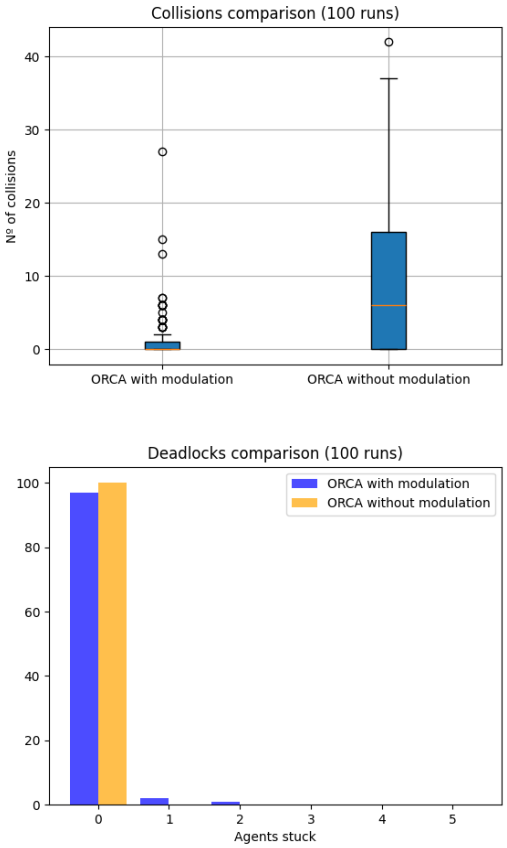

Figure 10: Collisions and deadlocks in the Home scenario between ORCA behavior with and without graph-based modulation.

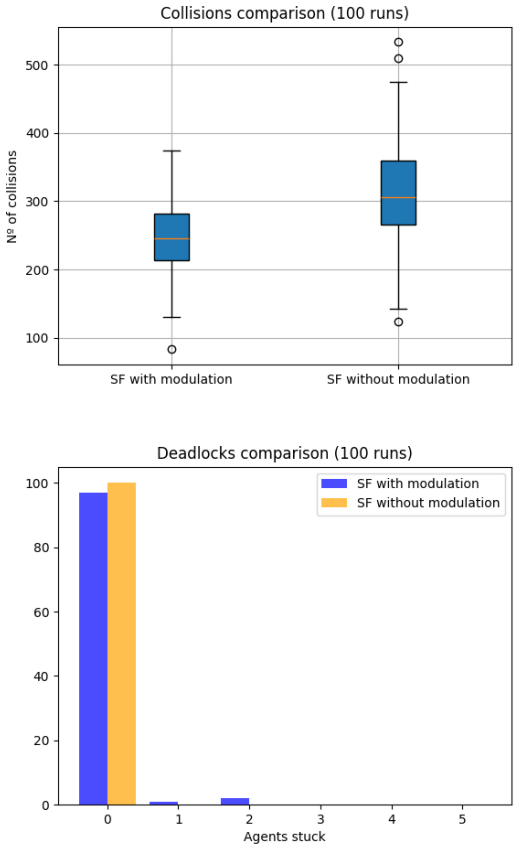

Figure 11: Collisions and deadlocks in the Home Scenario between SFM behavior with and without graph-based modulation.

to 6.42% when the modulation is applied. In general, we can say that the results obtained for ORCA are positive in the Home scenario because the decrease in the number of collisions is so significant that it compensates for the slight loss in terms of deadlocks and efficacy.

Finally, we repeated the analysis for the simulations using SFM as the baseline. Fig. 11 shows the summary of the results. In this case, the median number of collisions per simulation decreases from 305.5 to 245 when applying the graph-bawsed modulations. With respect to the number of deadlocks, we have similar results to those in the experiments with ORCA. When no modulation is applied, 100% of the simulations end without deadlocks, while when the modulation is applied, we have one simulation with one agent blocked and two with two agents blocked. The mean efficacy also goes down from 14.34% to 12.68% when applying the modulation. Once again, we can say that the improvement in the number of collisions is so significant that it compensates for the slight losses in terms of efficacy and the probability of deadlocks.

D Improvement in the Detection of Deadlock Agents

As previously noted, graph modulation significantly reduces the number of collisions between agents, especially in ORCA; however, it slightly worsens the number of agents in a deadlock state. This led us to consider the possibility that the NaveGround algorithm was either failing to adequately detect agents in a deadlock state or misinterpreting the undeadlock condition. NaveGround considers an agent to be deadlocked when it remains stationary. However, we encountered situations where, despite not being stationary, the agent moves very little within an area without heading in a fixed direction. This way, we consider the maximum speed for an agent to be classified as in a deadlock state to be 0.01 m/s.

For this reason, we devised a new parameter-dependent algorithm that, in particular, accurately detects agents in a deadlock state based on the criteria mentioned above. The algorithm takes as parameters a simulation, the minimum number of steps during which the agent must have a speed equal to or less than 0.01 m/s to be considered in a deadlock state,

and finally, whether modulation is applied or not.

The algorithm consists of checking the speed of all agents at each step. If the speed is equal to or less than the maximum considered speed, the counter for that agent is incremented by 1. If, in a step, the speed exceeds the maximum considered speed, the counter is reset to 0. If, at the end, an agent's counter is equal to or greater than the number of steps we consider to determine if the agent is in a deadlock state, it is classified as such. The modulation parameter, which indicates whether modulation is applied or not, is included because agents with modulation may be on standby, for example, to allow others to pass first. If the agent is on standby, it is not considered to be in a deadlock state.

We observed in Fig. 12 for the ORCA and SF behavior that with the new algorithm for counting deadlocks, more agents in a deadlock state are now being detected. Besides, we can observe that fewer agents are in such a deadlock state with modulation than without modulation.

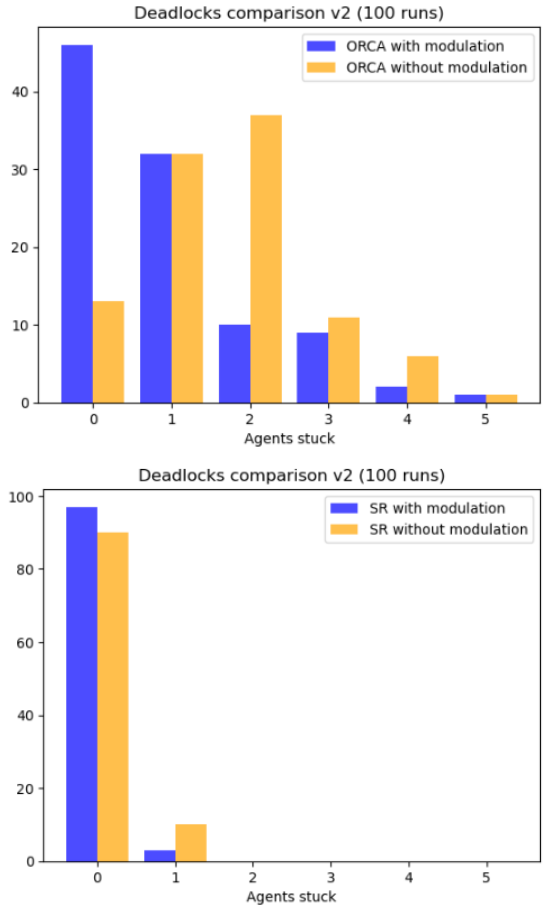

Figure 12: Deadlocks with the new algorithm in the Home Scenario with the ORCA and SF behavior with and without graph-based modulation.